

The NIH Biowulf Cluster – Scientific Supercomputing

Steven Fellini
sfellini@nih.gov

Susan Chacko
susanc@hpc.nih.gov

staff@hpc.nih.gov

NIH HPC Staff, CIT

Dec 11-12, 2017

Slides available at <https://hpc.nih.gov/training/>

Monday

- Introduction
- Why use Clusters/Biowulf?
- Administrivia: Accounts & Passwords, Connecting to Biowulf; Email
- Cluster Basics & Concepts
- Biowulf architecture & hardware configuration
- Environment modules
- Data Transfer & Storage, Snapshots
- Introduction to the batch system

Tuesday

- Swarm
 - Swarms of single-threaded, multi-threaded, auto-threaded and parallel jobs
- Partitions
- Walltimes, local disk, backfill scheduling, GPUs
- Monitoring your jobs
- Job dependencies
- Parallel jobs
- Containers

The NIH HPC Core Facility (Biowulf)

- Central scientific compute resource managed by NIH HPC staff (CIT)
- Funded by NIH Capital Investment Fund
- Available to all NIH intramural scientists
- Production facility: high availability, high data durability
- Large scale: 90,000+ processor cores; 14 Petabytes storage capacity
- Enabling research not otherwise possible
- General purpose scientific computing (not dedicated to any one application type)
- Cited by 380+ publications in 2016



NIH HPC Core Facility

hpc.nih.gov

Helix: Single shared system: 128 CPUs & 1 TB memory.
Sandbox for Linux practice, file management, lightweight computations, interactive graphics, specific licensed software (e.g. Comsol)

Felix (NIMH only)

Helixweb: web applications (*helixweb.nih.gov*)

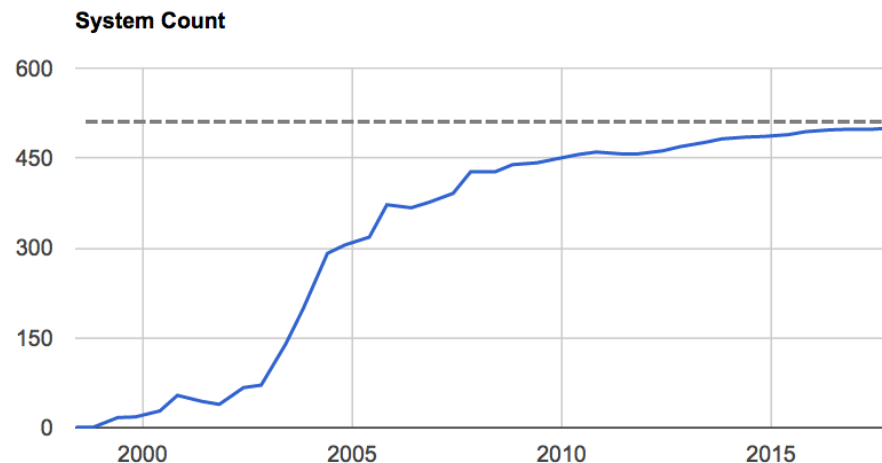
Sciware: run apps on your desktop (e.g. Matlab) (*sciware.nih.gov*)

Biowulf Cluster: High performance computing
For most computational jobs.

Biowulf & Helix run the Linux Operating System

RHEL/Centos 6, soon to be RHEL/Centos 7
100% of the Top500 supercomputers run Linux

OPERATING SYSTEM FAMILY / LINUX



List	Count	System Share (%)
Nov 2017	500	100
Jun 2017	498	99.6



(Nov 2017)

<http://top500.org>

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
65	Universitaet Mainz Germany	Mogon II - NEC Cluster, Xeon Gold 6130 16C 2.1GHz,	49,432	1,967.8	2,800.9	657
66	National Institutes of Health (NIH) United States	Biowulf - Apollo 2000 Gen 8/9, Xeon E5-2680v4/E5-2695v3 14C 2.4GHz, Infiniband FDR HPE	66,304	1,966.1	2,491.4	
67	Japan Atomic Energy Agency (JAEA) Japan	SGI ICE X, Xeon E5-2680v3 12C 2.5GHz, Infiniband FDR HPE	60,240	1,929.4	2,409.6	3,012

Why would you want to use Biowulf?

- Large numbers of jobs (bioinformatics on thousands of sequences)
- Large-memory jobs (whole genome assemblies)
- Parallel jobs with high-cpu demands (e.g., molecular dynamics on 1024 cores)
- Terabytes of data to process

... in other words, LARGE SCALE



Unsuitable for Biowulf...(or, why bother?)



- Small numbers of jobs
- One dependent job after the other
- Interactive jobs, graphics etc.

... in other words, desktops can be pretty powerful!

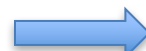
Class survey

- Level of Unix/Linux experience?
- HPC (Biowulf/Helix) account?
- Ever used a batch system?
- Your research application?

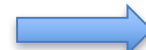
<http://hpc.nih.gov>



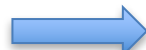
<http://hpc.nih.gov/apps>



User Guides




Announcements



HPC @ NIH

High-Performance Computing at the NIH

[Status](#) [Applications](#) [Storage](#) [User Guides](#) [User Dashboard](#) [How To](#) [About](#)




The NIH HPC group plans, manages and supports high-performance computing systems specifically for the intramural NIH community. These systems include **Biowulf**, a 90,000+ processor Linux cluster; **Helix**, an interactive system for short jobs; **Sciware**, a set of applications for desktops, and **Helixweb**, which provides a number of web-based scientific tools. We provide access to a wide range of computational applications for genomics, molecular and structural biology, mathematical and graphical analysis, image analysis, and other scientific fields.

Quick Links

- [System Status](#)
- [How To...](#)
- [Application/DB updates](#)
- [User Guides](#)
- [SquirrelMail](#)
- [Policies](#)
- [Training](#)
- [Contact Us](#)

Biowulf Utilization

Monday, December 4th, 2017



Last 24 hrs

71,121 jobs submitted	21 NIH Institutes
43,388 jobs completed	137 Principal Investigators
1,516,083 CPU hrs used	233 users

Announcements

(All)

- **SCHEDULE UPDATE:** Discontinuing SSH to Helix from Outside the NIH Network AND Helix Email to be Retired (Nov 29th 2017)
- **Reminder:** Python in HPC seminar Thursday Nov 30 (Nov 29th 2017)
- **Upcoming Biowulf classes** (Nov 27th 2017)
- **REMINDER:** Walk-In Consult with HPC staff Wed 15 Nov (Nov 14th 2017)
- **Walk-In Consult with HPC staff** Wed 15 Nov (Nov 8th 2017)
- **REMINDER:** Seminar today - Effective use of the Biowulf batch system and storage systems (Oct 30th 2017)
- **NIH Biowulf Seminars:** Making Effective use of the cluster (Oct 24th 2017)

Recent Papers that used Biowulf & HPC Resources

(All publications)

Natural Language Processing for Large-Scale Medical Image Analysis Using Deep Learning
Hoo-Chang Shin, Le Lu, Ronald M. Summers
in Deep Learning for Medical Image Analysis, Elsevier, (p405-421) (2017)

Excessive burden of lysosomal storage disorder gene variants in Parkinson's disease
Robak, LA; Jansen, IE; van Rooij, J et al.
Brain, DOI:10.1093/brain/awx285 (2017)

Insights into functions of the H channel of cytochrome c oxidase from atomistic molecular dynamics simulations
Sharma, V; Jambrina, PG; Kaukonen, M; Rosta, E; Rich, PR;
Proc. Natl. Acad. Sci. U.S.A., DOI:10.1073/pnas.1708628114 (2017)

Conventional and pioneer modes of glucocorticoid receptor interaction with enhancer chromatin in vivo
Johnson, TA; Chereji, RV; Stavreva, DA; Morris, SA; Hager, GL; Clark, DJ;
Nucleic Acids Res., DOI:10.1093/nar/gkx1044 (2017)

A joint model for multivariate hierarchical semicontinuous data with replications
Kassahun-Yimer, W; Albert, PS; Lipsky, LM; Nansel, TR; Liu, A;
Stat Methods Med Res, DOI:10.1177/0962280217738141 (2017)

Phosphorylated Calmodulin Promotes PI3K Activation by Binding to the SH2 Domains
Zhang, M; Jang, H; Gaponenko, V; Nussinov, R;
Biophys. J., DOI:10.1016/j.bpj.2017.09.008 (2017)

Prediction of Host-Pathogen Interactions for Helicobacter pylori by Interface Mimicry and Implications to Gastric Cancer
Guven-Maiorov, E; Tsai, CJ; Ma, B; Nussinov, R;
J. Mol. Biol., DOI:10.1016/j.jmb.2017.10.023 (2017)

Physiological and pathophysiological implications of PGE2 synthases in the kidney
Wang, J; Liu, M; Zhang, X; Yang, G; Chen, L;
Prostaglandins Other Lipid Mediat., DOI:10.1016/j.prostaglandins.2017.10.006 (2017)

Social media, RSS

<http://hpc.nih.gov/research>

HPC @ NIH - Contact -
Disclaimer - Privacy - Accessibility - CIT - NIH - DHHS - USA.gov

11

Accounts, passwords etc.

Accounts



No sharing of accounts!

Apply for an account at

https://hpc.nih.gov/nih/accounts/account_request.php

PI approval needed.

\$35/month flat charge – no additional usage fees

Accounts must be renewed each year.

Accounts

Helix	—	Same username (NIH login username)
	—	Same password (NIH login password)
Biowulf	—	Same /home area
Felix	—	Same /data area

Need not have the same shell, but we recommend 'bash' for both.

You may get locked due to inactivity on any system: you can unlock yourself at <https://hpc.nih.gov/dashboard>

(or send email to staff@helix.nih.gov)

Email

Mail from:

- Biowulf batch system
- Biowulf sysadmins
- Notifications
- Helix announcements



NIH email address as in NED

Cluster Basics & Concepts

Cluster Basics



computer = node = blade

=



cpus
memory
disk
network adapter

cluster of 13 nodes

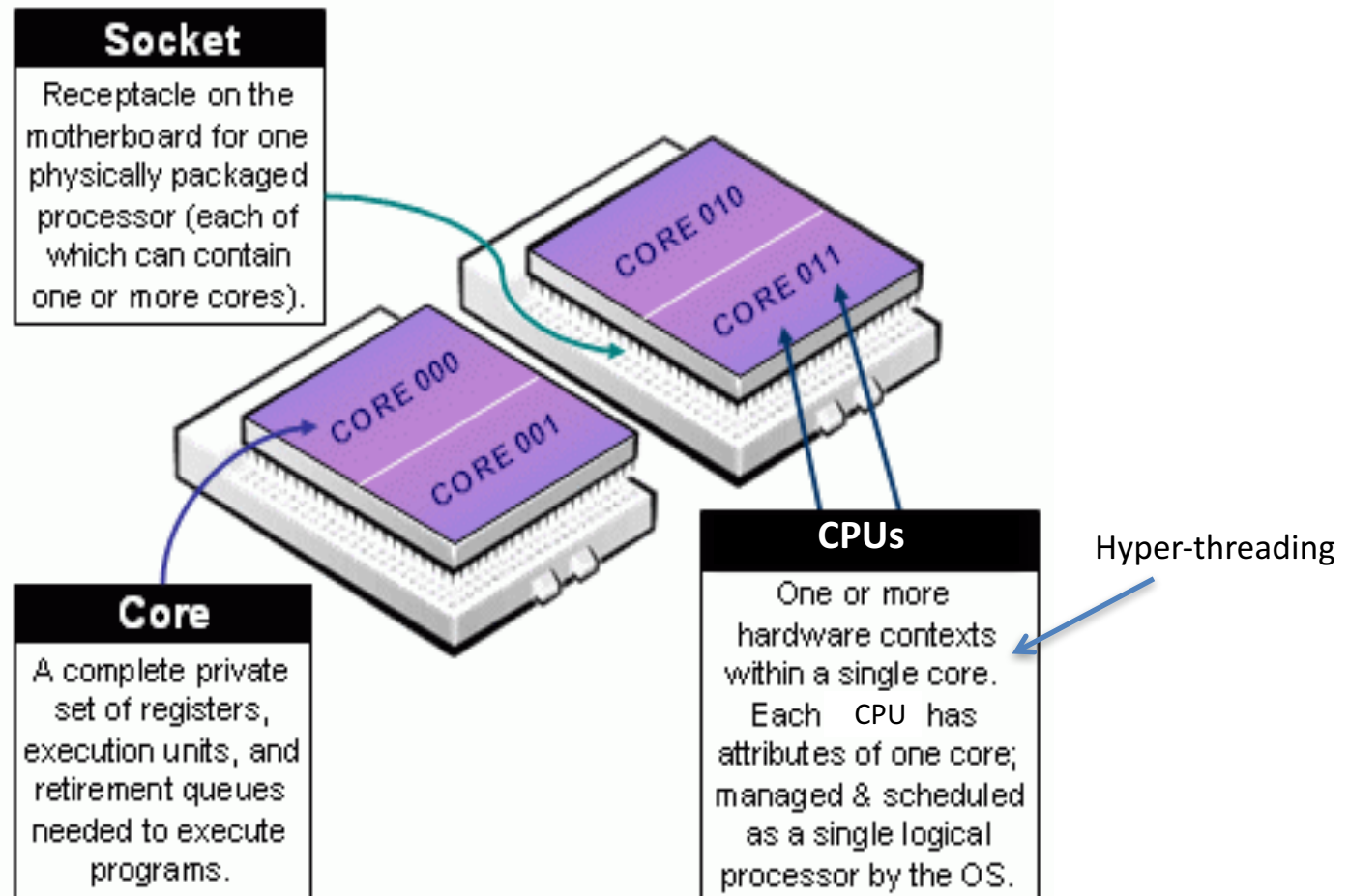


fileserver (disk storage)

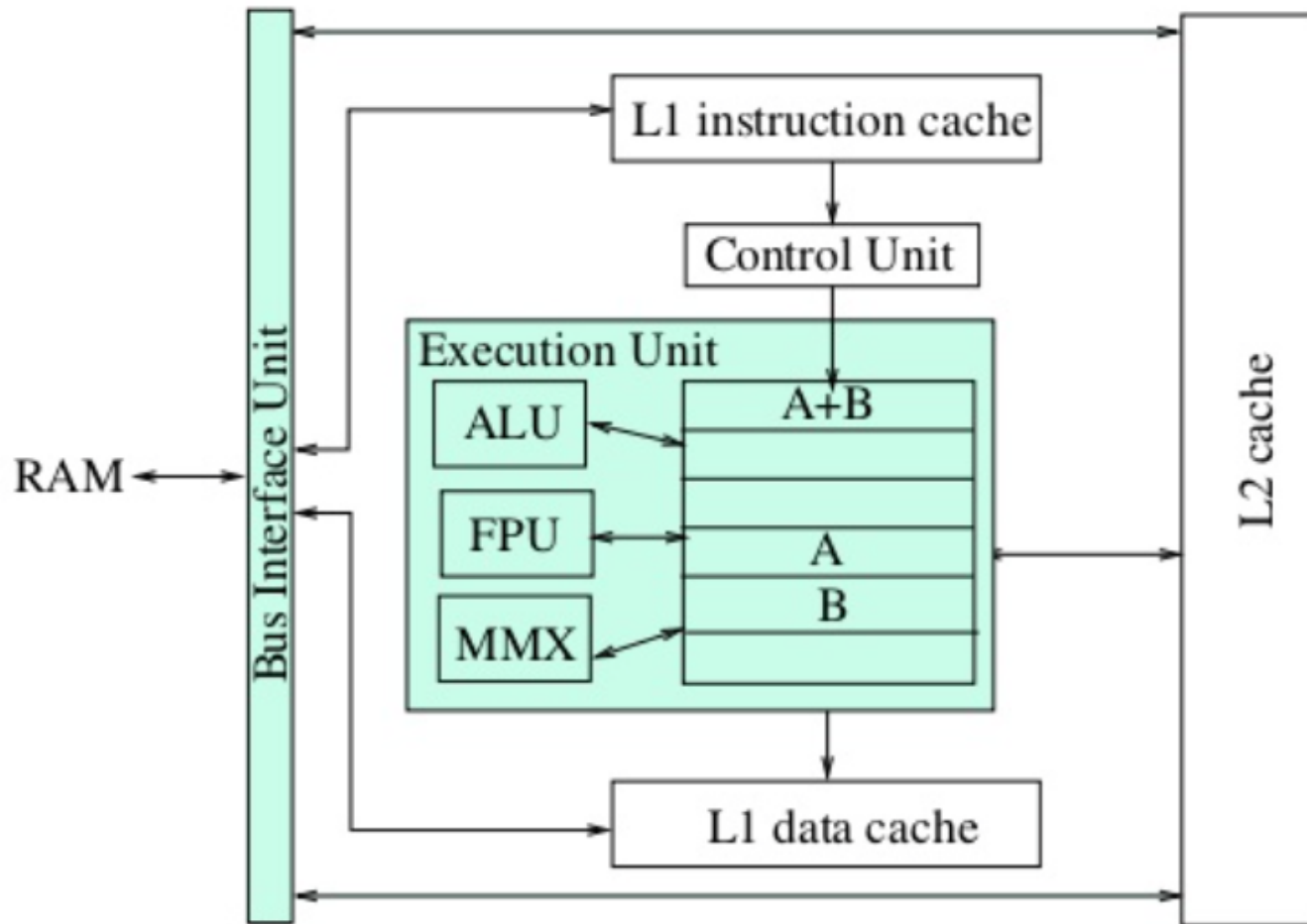


network switch

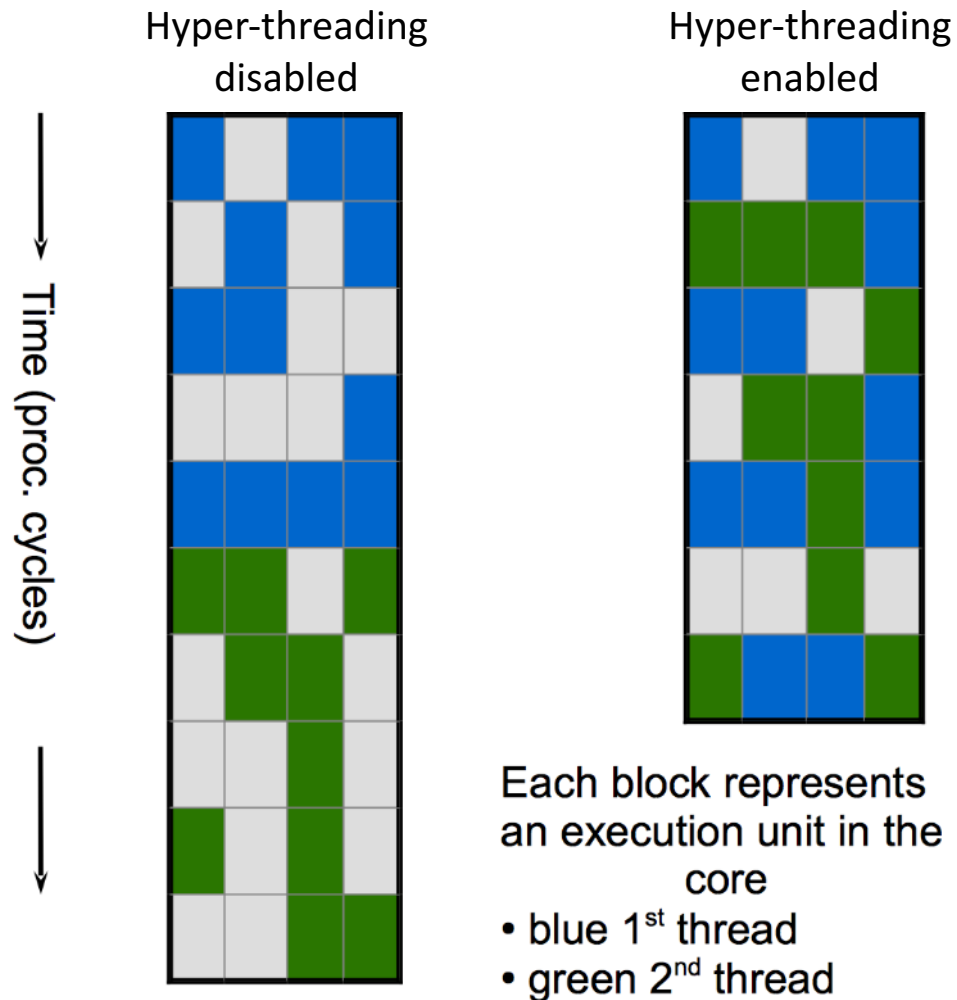
Hardware Terminology



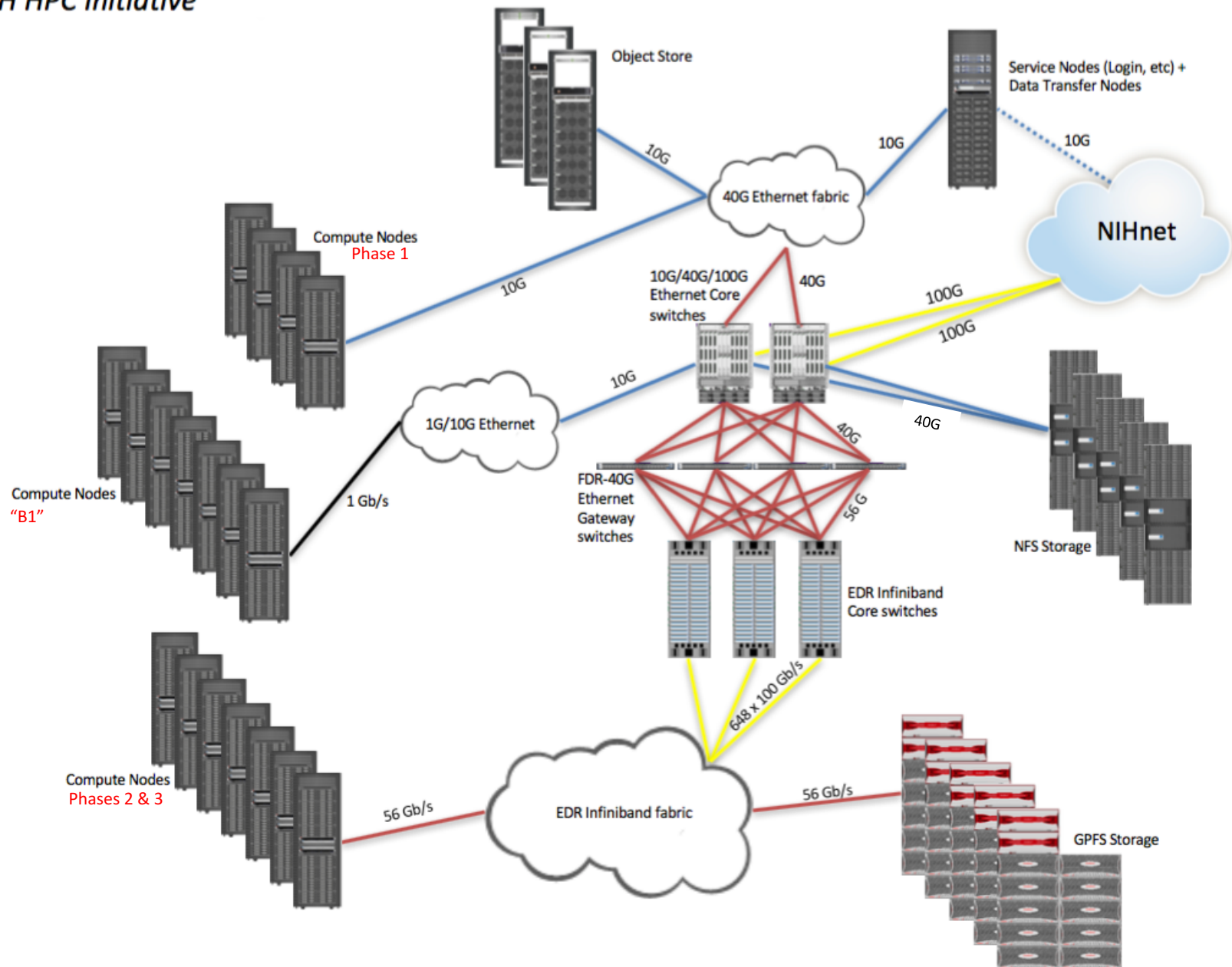
Processor Functional Units



Hyper-threading



NIH HPC Initiative



Compute Nodes (1/2)

# nodes	processor, cores per node	memory	network
1080	Intel E5-2680v4 28 x 2.4 GHz	256 GB	56 Gb/s FDR Infiniband
1080	Intel E5-2695v3 28 x 2.3 GHz	256 GB	56 Gb/s FDR Infiniband
324	Intel E5-2650v2 16 x 2.6 GHz	128 GB	10Gb/s Ethernet
192	Intel E5-2650v2 16 x 2.6 GHz	64 GB	56 Gb/s FDR Infiniband 10Gb/s Ethernet
72	Intel E5-2696v4 28 x 2.3 GHz 2 x nVIDIA K80	256 GB	56 Gb/s FDR Infiniband
24	Intel E5-2650v2 16 x 2.6 GHz 2 x nVIDIA K20x	128 GB	56 Gb/s FDR Infiniband 10Gb/s Ethernet

Compute Nodes (2/2)

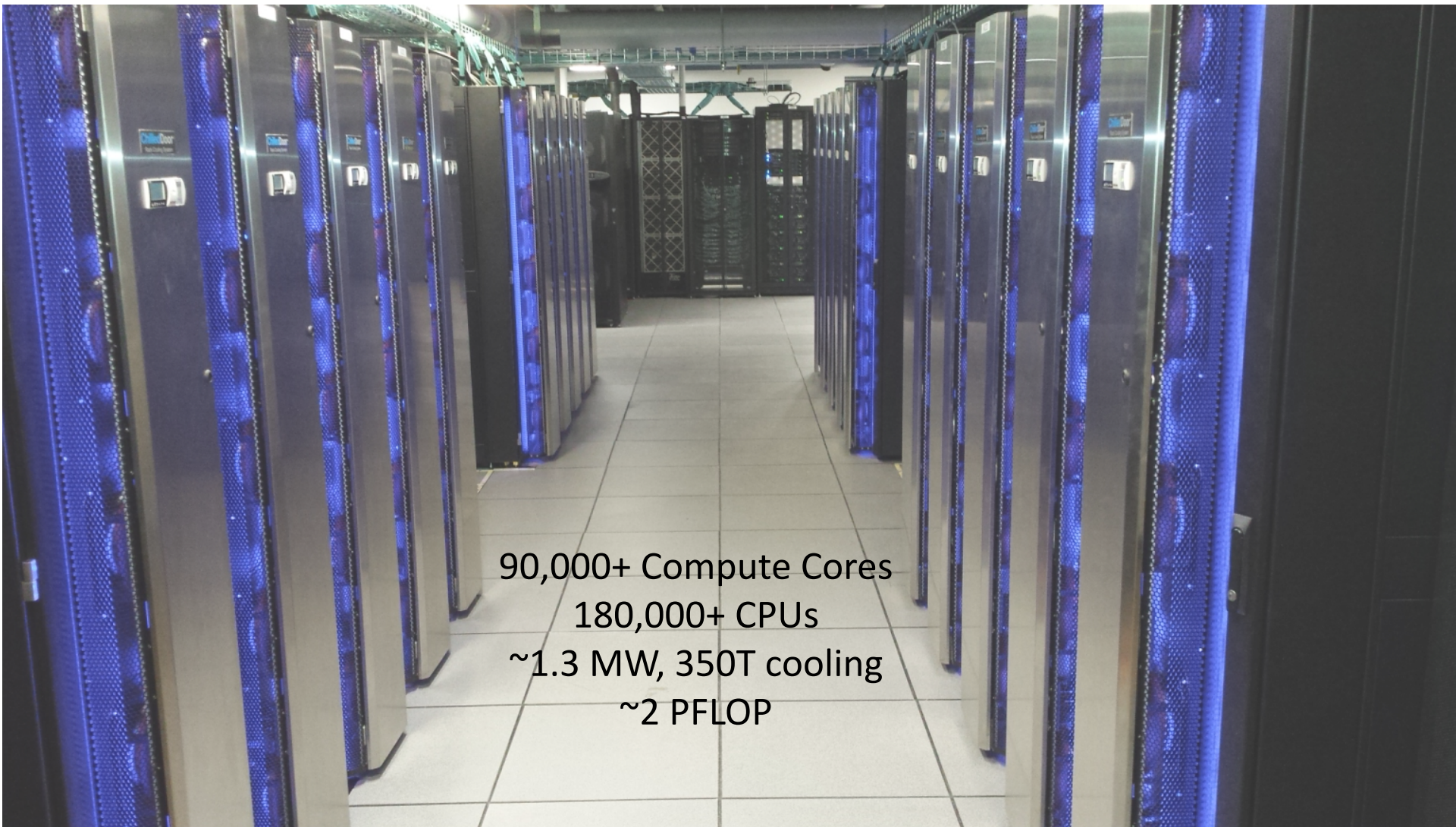
# nodes	processor, cores per node	memory	network
4	Intel E7-8860v4 72 x 2.2 GHz	3.0 TB	56 Gb/s FDR Infiniband
20	Intel E7-8860v4 72 x 2.2 GHz	1.5 TB	56 Gb/s FDR Infiniband
4	Intel E5-4620v2 32 x 2.6 GHz	1 TB	10Gb/s Ethernet
224 (-)	Intel E5-2670 16 x 2.6 GHz	128 & 64 GB	1 Gb/s Ethernet
362 (-)	Intel X5660 12 x 2.8 GHz	24 GB	1 Gb/s Ethernet
302 (-)	Intel E5-5550 16 x 2.67 GHz	24GB	1 Gb/s Ethernet

Buy-In Nodes

# nodes	processor, cores per node	memory	network	funding
160	Intel E5-2695v3 28 x 2.3 GHz	256 GB	56 Gb/s FDR Infiniband	CCR (NCI)
24	Intel E5-2695v3 28 x 2.3 GHz 2 x nVIDIA K80	256 GB	56 Gb/s FDR Infiniband	CCR (NCI)
86	Intel E5-2650v2 16 x 2.6 GHz	128 GB	10Gb/s Ethernet	NIDDK
96	Intel E5-2650v2 16 x 2.6 GHz	128 GB	10Gb/s Ethernet	CCR (NCI)
64	Intel E5-2650v2 16 x 2.6 GHz	128 GB	10 Gb/s Ethernet	NIMH

- Priority given to funding Lab/IC for 3 years
- Available via *quick* queue to all NIH users

NIH Biowulf Stats



90,000+ Compute Cores
180,000+ CPUs
~1.3 MW, 350T cooling
~2 PFLOP

Hands-on

Connecting to Biowulf

Macs/Linux: `ssh your-username@biowulf.nih.gov`

Windows: open a putty session to `biowulf.nih.gov`

- Use your own username/password.
- If you don't have an account, ask the instructor for a student account.

Use of Biowulf's login node

- Submitting jobs
- Editing/compiling code
- File management
- File transfer
- **Brief** testing of code or debugging (under 20 minutes cpu time)





Do not run compute jobs of any kind on the login node!

root

To: [REDACTED]@biowulf2.nih.gov

No compute intensive processes on Biowulf login node

Compute or large memory jobs are not allowed on the Biowulf login node. Such jobs can hang the login node and/or seriously affect all users. If you persist in running such jobs, your account will be locked.

Please use batch or allocate an interactive node if needed. Here is how easy it is to get a dedicated interactive session:

% sinteractive

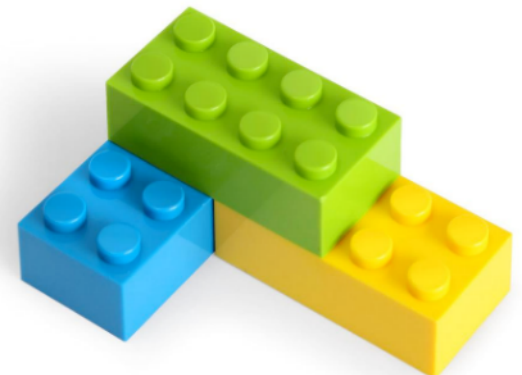
Process terminated:

PID	USER	NI	VSZ	RSS	S	%CPU	%MEM	TIME	COMMAND
23583	[REDACTED]	0	5973256	5463320	R	96.9	4.4	00:01:44	bedtools sort -i KZ946_CLP.bed



Environment Modules

- Dynamically set up environments (paths, library locations, etc.) for different applications
- One easy command for setup (not shell-dependent)
- <https://hpc.nih.gov/apps/modules.html>

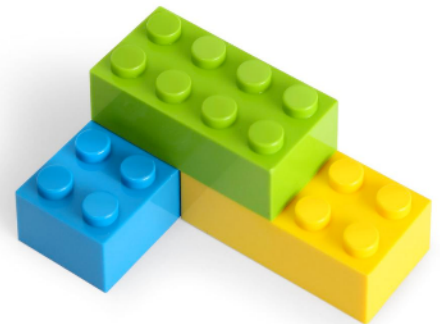


What modules exist?

module avail

module avail [appname|string|regex]

module -d (show only default versions)



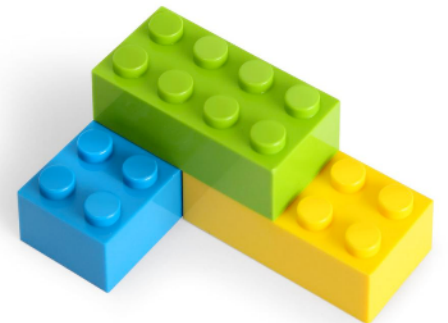
Load a module

module load appname

module load appname/version

See loaded modules

module list



Unload a module

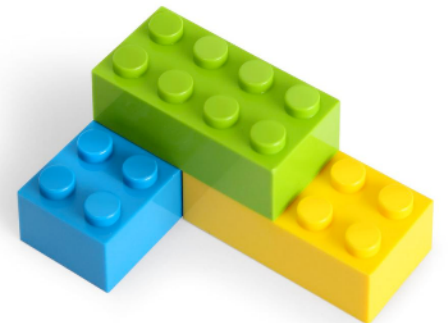
`module unload appname`

`module purge` (unload all modules)

Personal modules

`module use --append ~/mymodules`

`module use --prepend ~/mymodules`



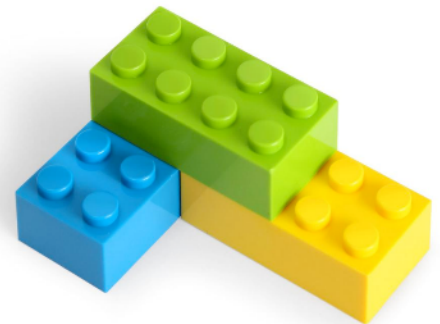
Examine a module

`module whatis appname`

`module help appname`

`module [display|show] appname`

`module [display|show] appname/version`



Convenient shorthand

```
ml
```

module list

```
ml STAR
```

module load STAR

```
ml -STAR
```

module unload STAR

```
ml STAR -bwa
```

load STAR and unload

```
ml cmd
```

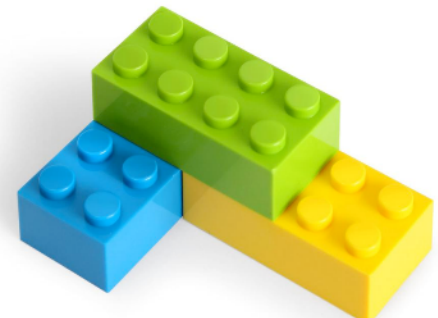
bwa

module *cmd*

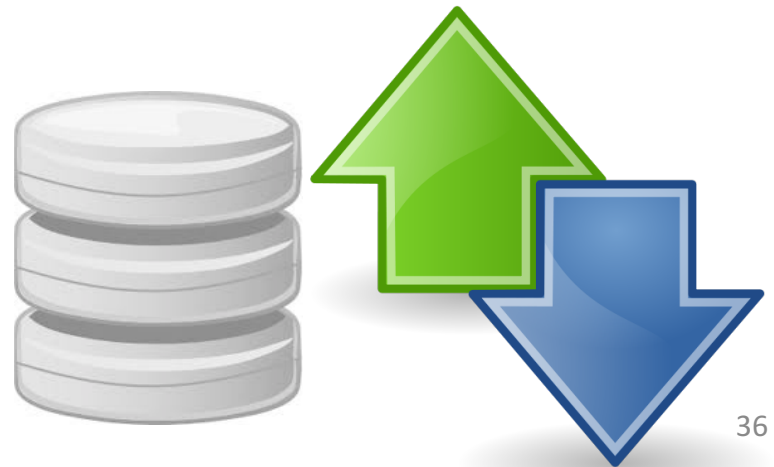
Hands-on (modules)

```
$ module avail -d
$ module avail star
$ echo $PATH
$ module load STAR
$ echo $PATH
$ module list
$ ml -STAR
$ echo $PATH
$ module display bedtools
```

Note how your \$PATH changes when a module is loaded or unloaded.



Data Storage and Data Transfer



PII & PHI data

NIH HPC users are **forbidden** from transmitting or storing **any** Personally Identifiable Information (PII, e.g. patient data containing names or social security numbers) or Protected Health Information (PHI) data anywhere on the NIH HPC systems, including their /home, /data, and any group (shared) /data directories.

Controlled access data such as dbGaP data can be stored on the systems, but it is the responsibility of the user to fulfill all requirements of the agreement with the database provider.

<https://hpc.nih.gov/policies>

Disk storage

	Location	Creation	Backups?	Amount of space	Accessible from (*)
/home	Network filesystems	with HPC account	snapshots & tape	8 GB (quota)	B, H, C

(*) H = helix, B = biowulf login node, C = biowulf compute nodes



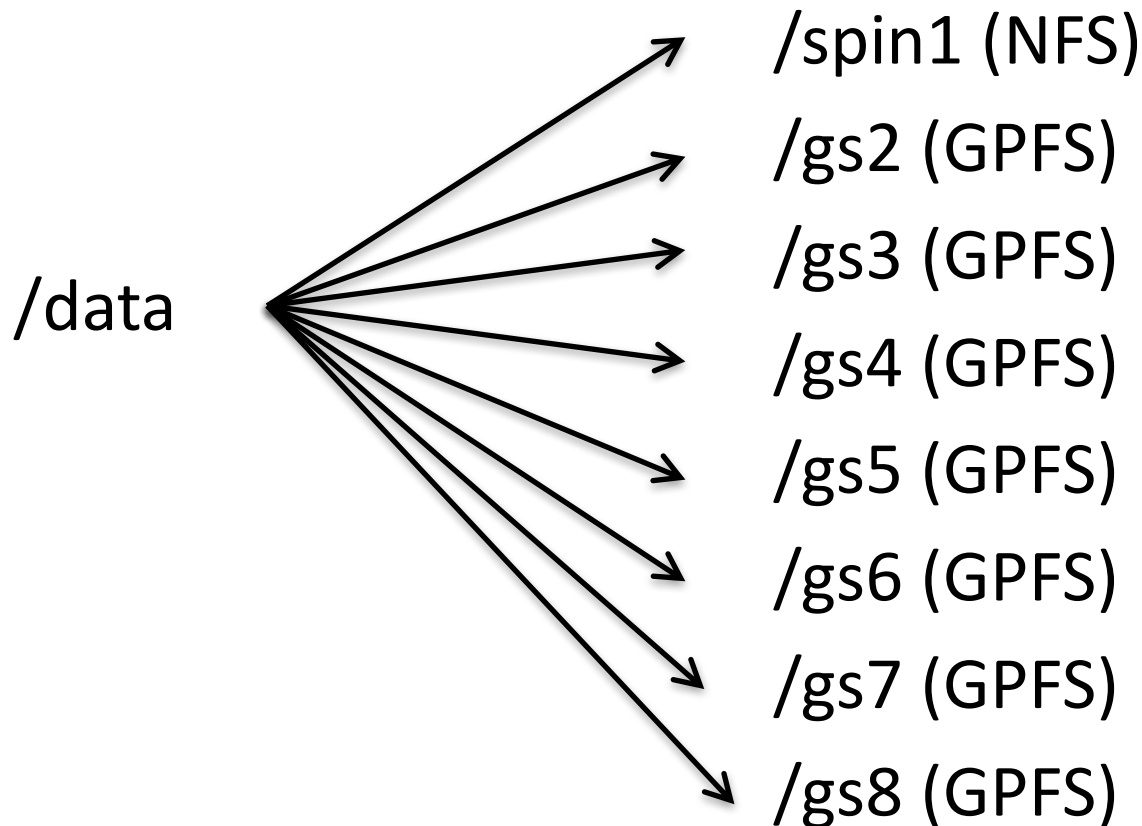
Most /data directories are on high performance
parallel filesystems (GPFS)
(14 PB and climbing)



Biowulf FileSystems



Use /data/username, **not** /gs1/users/username



How to check quota and usage

\$ **checkquota**

Mount	Used	Quota	Percent	Files	Limit
/data:	125.8 GB	500.0 GB	25.16%	16529	31129581
/home:	2.4 GB	8.0 GB	30.37%	52888	n/a
ObjectStore Vaults					
teacher:	44.4 GB	465.7 GB	9.00%	n/a	n/a

Quotas and shared directories

- Quota increases for /data
- Shared data directory requests

hpc.nih.gov/dashboard

-> 'disk usage' tab



Hands-on (disk storage)

Simple commands

```
ls -la /home/$USER
```

(check contents of your /home area)

```
ls -l /data/$USER
```

(check contents of your /data area)

```
checkquota
```

(check your disk space usage/allocation)

```
mkdir /scratch/$USER
```

(create a directory in /scratch)

```
cp /home/$USER/file /scratch/$USER/
```

(copy a file to your /scratch dir)

Copy the scripts/data for this class to your own area

```
hpc-classes biowulf
```

(Copy takes about 4 mins – be patient. You should now have a directory /data/\$USER/hpc-classes/biowulf containing 3.8 GB of data)



Snapshots

/home	6 hourly, 6 nightly, 8 weekly
/data	2 daily, 1 weekly
/scratch	no snapshots



Snapshots are not backups



Hands-on (snapshots)

```
cd  
ls -a  
cd .snapshot  
ls  
cd Nightly.[date]  
cp somefile /home/$USER  
  
cp -p somefile /home/$USER
```

(cd to your /home area)

(can you see the .snapshot directory?)

(cd to the snapshot in that area)

(see what snapshots are available)

(cd to a specific snapshot)

(copy a file from that snapshot to /home)

(to preserve timestamps and permissions)



Best Practices for Storage

BAD

Submitting a swarm without knowing how much data it will generate

Directory with 1 million files

Each collaborator having a copy of data on Biowulf

Using Biowulf storage for archiving.

GOOD

Run a single job, sum up the output and tmp files, and figure out if you have enough space before submitting the swarm.

Directories with < 5,000 files.

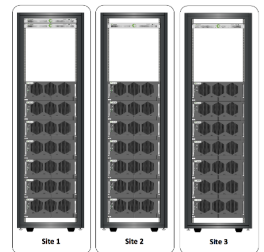
Ask for a shared area and keep shared files there to minimize replication.

Move unused or old data back to your local system.



Object Storage

- “Web Scale” storage
 - Scalable to billions of objects
 - Highly durable (Erasure encoding, dispersed over multiple sites)
- Different from file based storage systems
 - Objects are accessed by **NAME**, not **PATH**
 - Accessed via RESTful API: list, put, get, delete
 - Completely flat name space
 - No concept of directories, but “/” is a valid character in object names
 - Various toolkits are available to manipulate objects (for instance, obj command) (or write your own)
 - Data and metadata are stored together with the object

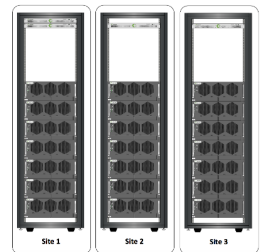


Use-cases for object storage

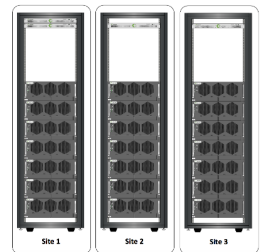
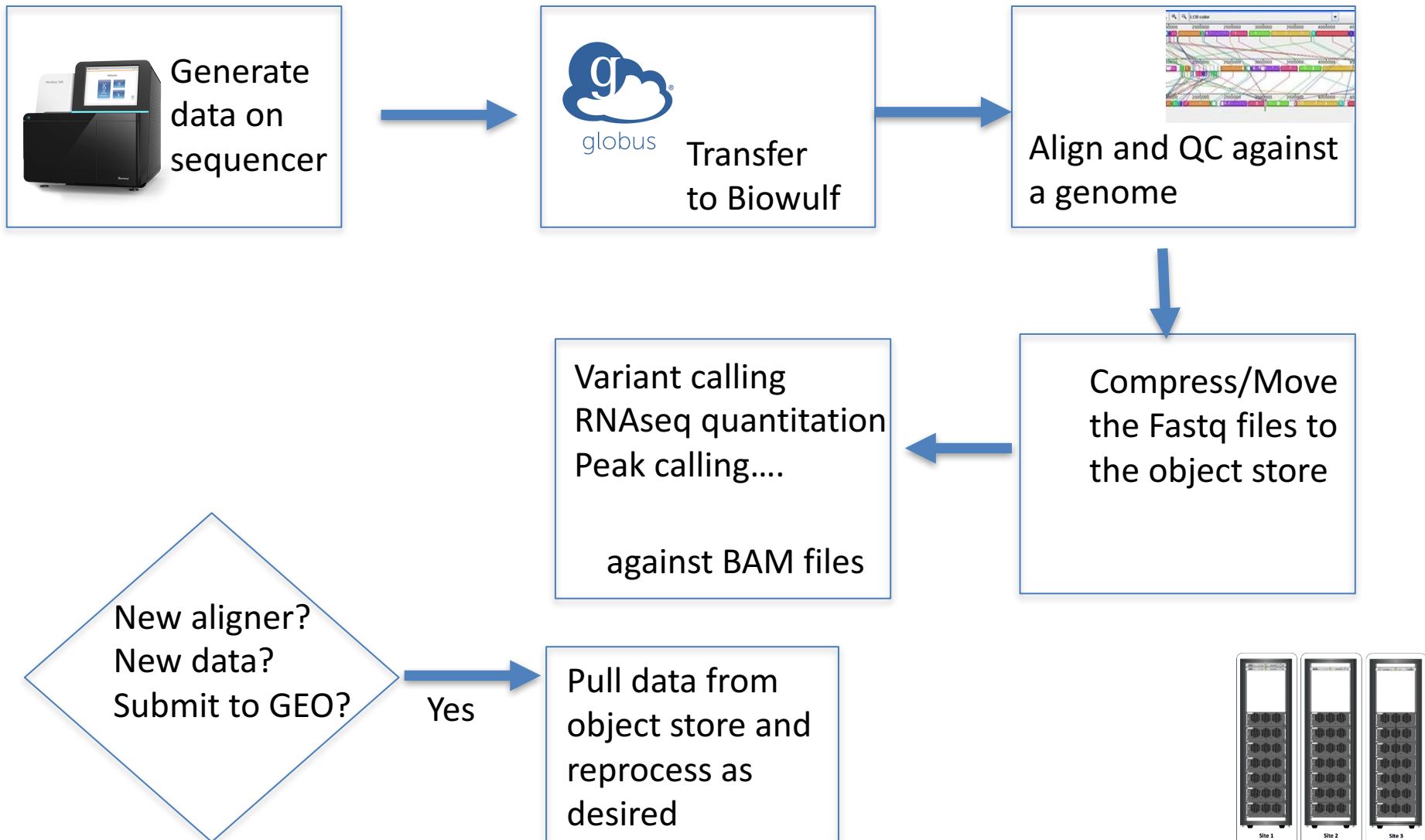
- Read-intensive workloads
 - Object storage is much more efficient at reading than writing.
 - An **entire** object has to be re-written for each change
 - Computationally expensive to process and disperse the data
 - Lots of over-writing
- Static data
 - “Write-once, read-many”
 - Data that doesn’t change often, but still used
 - E.g. reference genomics files
- Limited archive?
 - “Park” your processed data until the paper is published

To request an object store allocation...

https://hpc.nih.gov/nih/object_request.html



Typical Genomics Workflow



Upcoming Class...

The NIH HPC object store

Date: Jan 9, 2018

Time: 9 am - 1 pm

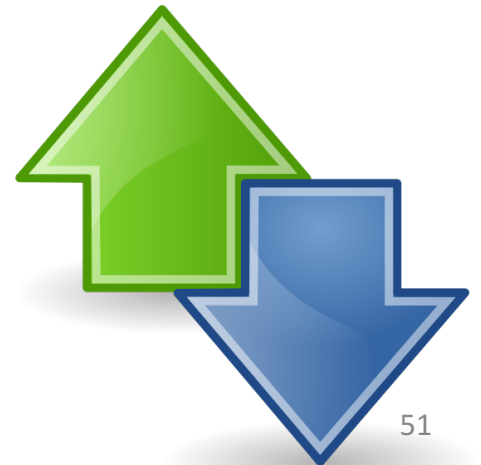
Instructor: Tim Miller (NIH HPC staff)

Location: Bldg 12A, Rm B51.

Transferring files



<https://hpc.nih.gov/docs/globus.html>



Transferring files

<http://hpc.nih.gov/docs/transfer.html>

- Map your Helix /home or /data area on your desktop (Helixdrive)
- GUI File transfer clients (WinSCP etc.)
- Commandline (`scp`, `sftp`, `bbcp`)
- Remember: Windows files have Ctrl-M characters as linefeeds (`dos2unix` to fix the file)



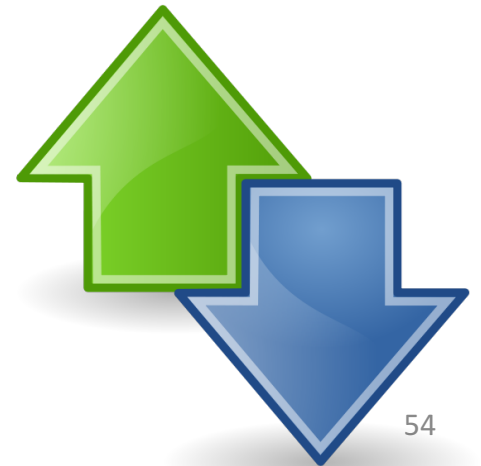
Web Proxy

- Allows limited Internet access *from compute nodes*
- Transparent for programs using HTTP_PROXY, HTTPS_PROXY, RSYNC_PROXY or FTP_PROXY environment variables
- Transfers through Data Transfer Nodes
- wget, curl, lftp, rsync, git clone
- (NCBI SRAToolkit, Hisat)
- Aspera (ascp) transfers: contact us



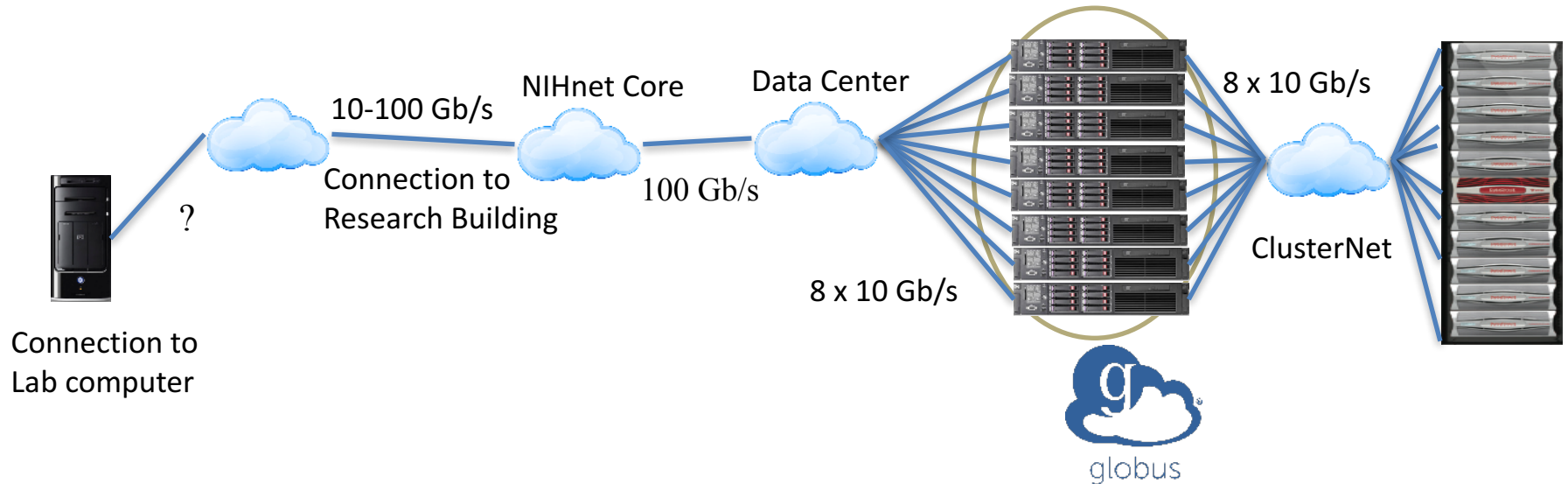
Sharing data with collaborators

https://hpc.nih.gov/docs/sharing_data.html



High Performance Data Transfers

- Data Transfer Nodes
- gridFTP
- Single Globus endpoint: *nihhpc#globus*



The Batch System: SLURM



Slurm is the workload manager on about 60% of the **TOP500** supercomputers,

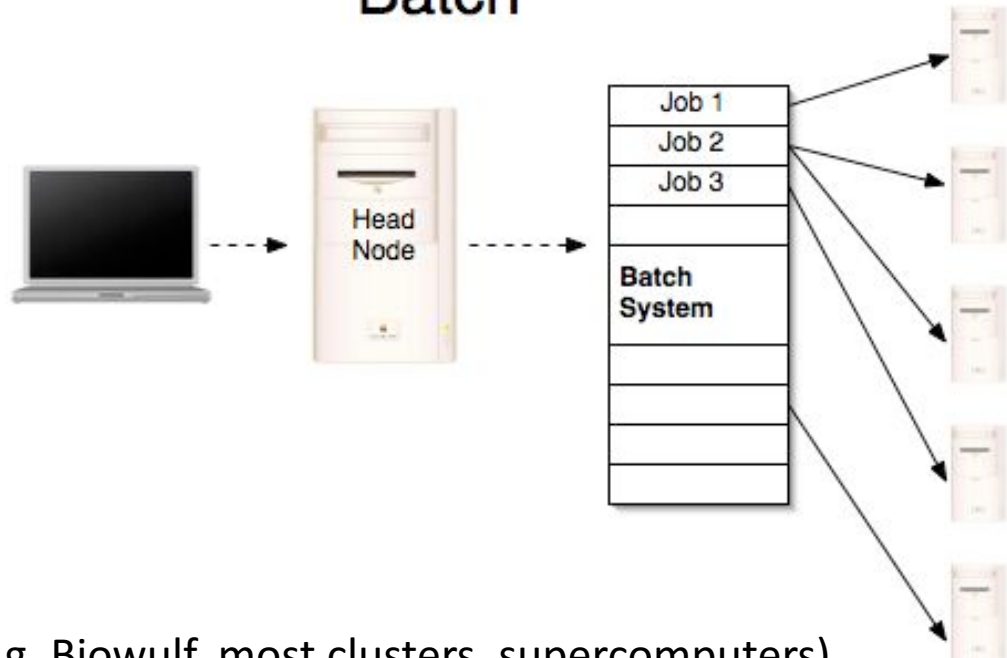
Concepts 1: Interactive vs. Batch

Interactive

(e.g. your desktop or Helix)

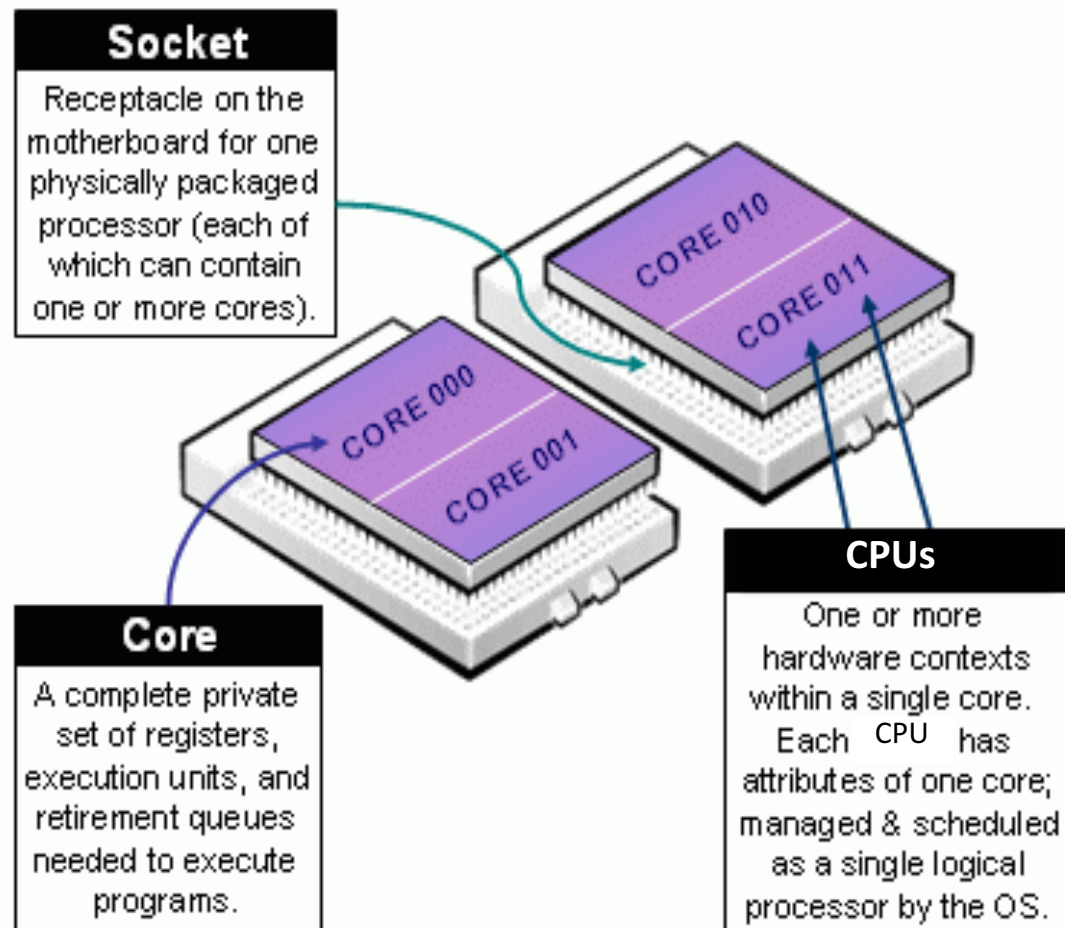


Batch

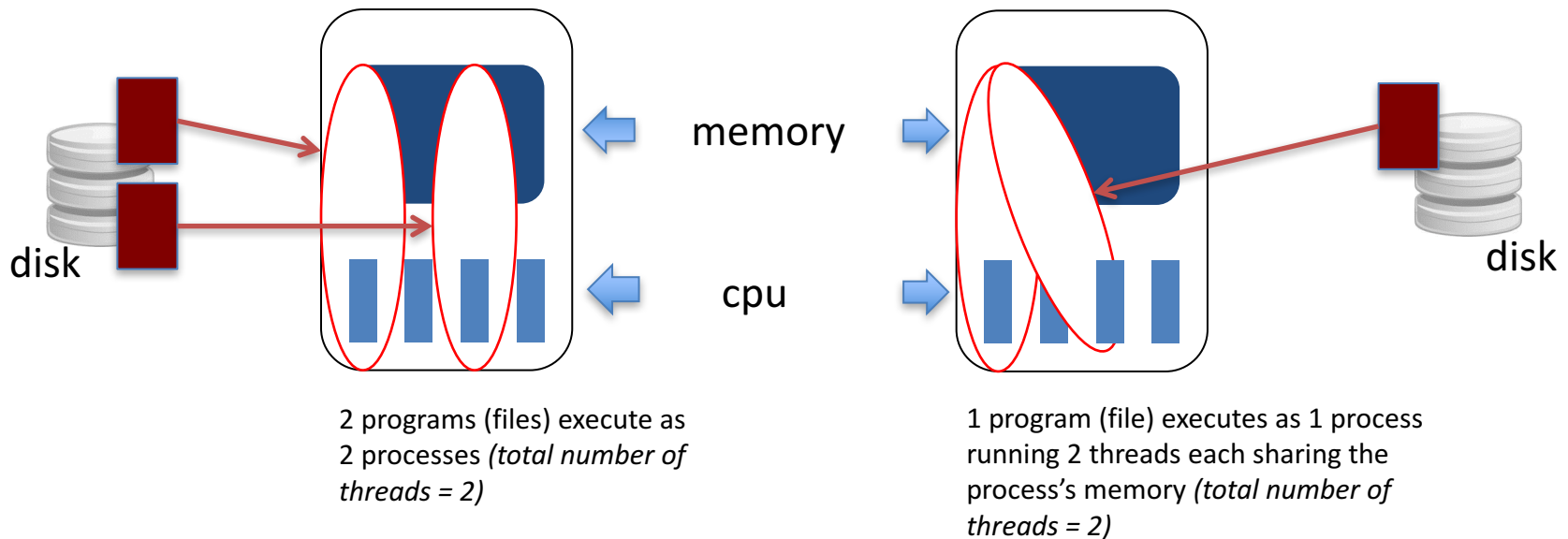


(e.g. Biowulf, most clusters, supercomputers)

Concepts 2: SLURM Hardware Terminology



Concepts 3: Processes & Threads



“A process has a self-contained execution environment; each process has its own memory space. Processes are often seen as synonymous with programs or applications.”

“Threads exist within a process — every process has at least one. Threads share the process's resources, including memory and open files.”

SLURM calls processes “tasks”

Concepts 4: Binding

SLURM uses the Linux kernel to bind processes to cores

Example:

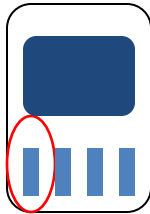
- 16-core node (32 CPUs)
- Submit jobs that allocate 1 core (2 CPUs)

Job 1: Application runs 1 thread, Result: 1 thread runs on 1 core (50% utilization of the core)

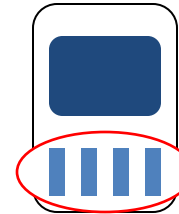
Job 2: Application runs 16 threads, Result: 16 threads run on 1 core (800% overloading of the core!)

Kinds of Jobs on a Cluster

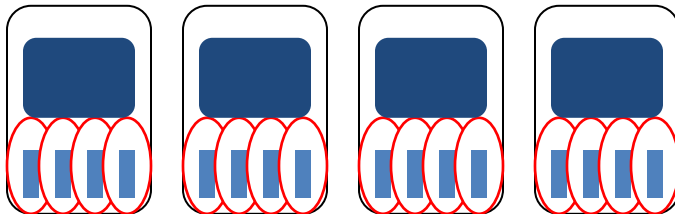
- Single-threaded apps



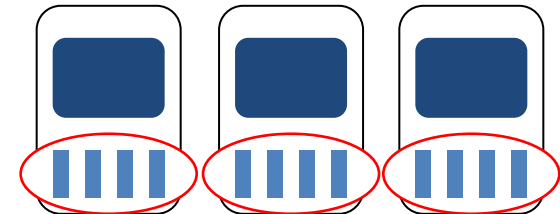
- Shared-memory parallel apps (multi-threaded)



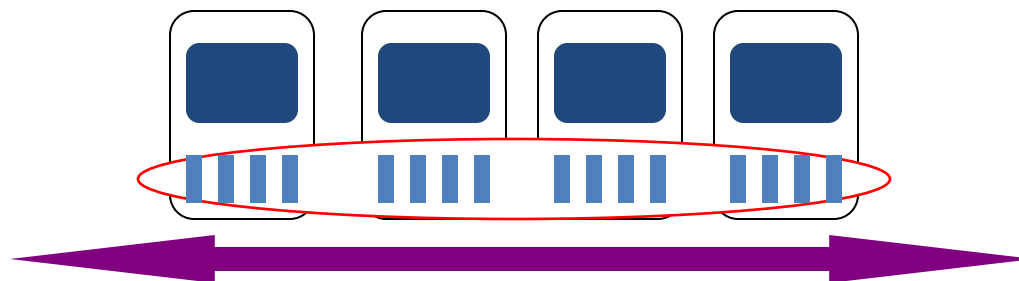
- Swarm of single-threaded processes



- Swarm of multi-threaded processes



- Distributed-memory parallel apps



Jobs on the Biowulf Cluster

- Most (almost all) jobs should be run in batch
- Two ways to submit batch jobs
 - sbatch
 - swarm
- Default compute allocation = 1 physical core
= 2 CPUs in Slurm notation
- Default Memory Per CPU = 2 GB
Therefore, default memory allocation = 4 GB

Submitting a simple serial job

```
#!/bin/bash
#
# this file is myjob.sh
#
#SBATCH --job-name MyJob
#SBATCH --mail-type BEGIN,END
#
myprog -n 100 < infile
```

```
% sbatch myjob.sh
49455953
```

jobid

NOTES:

- (1) sbatch exports your shell environment to the batch job
- (2) Command line options can also be used as directives within script (#SBATCH)
- (3) Identify the *jobid* when reporting problems to staff

Output/error files

Slurm combines stdout and stderr into a single file called `slurm-<job number>.out`

Default location: submission directory

Can be reassigned with

```
--error /path/to/dir/filename  
--output /path/to/dir/filename
```

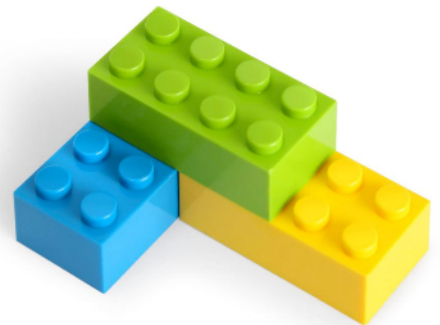
Using modules in batch scripts

```
#!/bin/bash
```

```
# load the latest version of bsoft  
module load bsoft
```

```
# load a particular version of bsoft  
# module load bsoft/1.8
```

```
cd /data/user/myjob  
....some bsoft command.....
```



Other slurm commands

Command	Purpose	Pros & Cons
squeue	Information about jobs	Lots of options, odd syntax
scancel	Delete job(s)	Only way to cancel your jobs!
sacct	Job accounting for completed jobs	Useful to get a list of jobs you submitted recently (default is 'today'), odd syntax
sjobs (Biowulf utility)	Info about running and pending jobs	Informative but slow

Deleting Batch Jobs

```
scancel <jobid> <jobid> ...
```

Options

- user
- name
- state

Multiple ways of setting batch parameters

```
sbatch --job-name=MyJob jobscript
```

Command-line parameter

```
export SBATCH_JOB-NAME=MyJob
```

Environment variable

```
#!/bin/bash
#
# this file is myjob.sh
#
#SBATCH --job-name MyJob
#SBATCH --mail-type BEGIN,END
#
myprog -threads=8 < infile
```

Batchscript directive

Command-Line Parameter >

Environment Variable >

Batch script Directive

SUMMARY: Variables set by SLURM when a batch job runs

Variable	Value
<code>\$SLURM_NODELIST</code>	Nodes allocated to the job
<code>\$SLURM_NTASKS</code>	No. of 'tasks' (usually MPI processes) for this job
<code>\$SLURM_CPUS_PER_TASK</code>	Number of CPUs allocated to each task. Often used for multithreaded processes
<code>\$SLURM_JOB_ID</code>	Job ID of this job

Allocating compute resources with *sbatch*

- --mem
- --mem-per-cpu
- --ntasks
- --cpus-per-task
- --ntasks-per-core
- --gres
- --time

Special case only...

- --exclusive
- --constraint

Single-threaded app

Required		Command	Allocated	
CPU	Memory		CPU	Memory
1	< 4 GB	<code>sbatch jobscript</code>	2	4 GB
1	<i>M</i> GB (<i>M</i> > 4)	<code>sbatch --mem=<i>M</i>g jobscript</code>	2	<i>M</i> GB



Note: *jobscript* must always be the last parameter on the sbatch command line

Hands-on

Submitting a single batch job

```
cd /data/$USER/hpc-classes/biowulf/plink
more plink.bat
sbatch [--partition=student] plink.bat
queue -u $USER
sjobs
```

- Which node is your job running on?
- Find the output file: Examine it for errors.
- How many cores/CPU's were allocated?
- How many cores/CPU's were used?

Multi-threaded app

Required		Command	Allocated	
CPU	Memory		CPU	Memory
C threads on C CPUs	M GB	<code>sbatch --cpus-per-task=C --mem=Mg jobscript</code>	C (rounded up to nearest 2)	M GB

Default memory allocation for **C** cpus would be (**C***2) GB

Multi-threaded app

Use `$SLURM_CPUS_PER_TASK` inside the script

```
#!/bin/bash

cd /data/mydir
module load hmmer
hmmsearch --cpu $SLURM_CPUS_PER_TASK \
  globins4.hmm /fdb/fastadb/nr.aa.fas
```

```
#!/bin/bash
cd /data/mydir
module load mapsplice
mapsplice.py -1 Sp_ds.10k.left.fq -2 Sp_ds.10k.right.fq \
  -p $SLURM_CPUS_PER_TASK
```

Hands-on (multi-threaded Tophat)

```
cd /data/$USER/hpc-classes/biowulf/bowtie  
sbatch [--partition=student]--cpus-per-task=8 --mem=5g tophat.bat
```

Use 'sjobs' to see if the job is running.

Look at the tophat.bat script to see how the number of CPUs (threads) is specified to tophat.

Why is this important?

Auto-threading apps

```
sbatch --exclusive --cpus-per-task=32 --constraint=cpu32 jobscript
```

will give you a node with 32 CPUs.

Interactive compute Jobs

Reasons to use interactive nodes

- Testing/debugging cpu-intensive code
- Pre/post-processing of data
- Graphical application
- GUI interface to application

Reasons *not* to use interactive nodes

- Easier than setting up a batch job

Interactive compute jobs

```
[biowulf ~]$ sinteractive  
salloc.exe: Granted job allocation 22374  
[cn0004 ~]$ exit  
exit  
salloc.exe: Relinquishing job allocation 22374
```

```
[biowulf somedir]$ sinteractive --cpus-per-task=4 --mem=8g  
salloc.exe: Granted job allocation 22375  
[cn0004 somedir]$
```

```
[biowulf ~]$ sinteractive --constraint=ccr ...
```

(do not need `--partition=student`)

Note: your environment will be exported to the job by default.

Interactive compute jobs

- The maximum allowable number of cores allocated to interactive jobs is 32 (64 CPUs).
- By default, the job will be terminated after 8 hours. You may extend this time to a maximum of 36 hours by using the "--time" option to the `sinteractive` command.

Hands-on (interactive batch jobs)

```
$ sinteractive
```

Once you are logged into a node:

```
$ cd /data/$USER/hpc-classes/biowulf/freebayes  
$ module load freebayes  
$ freebayes -f genome.fasta input.bam  
$ exit
```

(no need for `-partition=student`, since you are submitting to the interactive partition)

Requeue Job?

```
sbatch --requeue
```

default

or

```
sbatch --no-requeue
```

Tuesday

- Swarm
 - Swarms of single-threaded, multi-threaded, auto-threaded and parallel jobs
- Partitions
- Walltimes, local disk, backfill scheduling, GPUs
- Monitoring your jobs
- Job dependencies
- Serial->swarm demo || Parallel jobs
- Containers

Job Arrays

A group of independent jobs, largely identical.

Examples:

- Processing 1000 images with AFNI
- Running Blast with 500 query sequences
- Aligning 50 genome fragments against hg19

Q: how to generate, edit or delete 1000 batch scripts/jobs?

A: job arrays

Job Arrays

```
#!/bin/bash

cd /data/$USER/mydir
tophat genome_index ${SLURM_ARRAY_TASK_ID}.fastq \
> ${SLURM_ARRAY_TASK_ID}.out
```

```
sbatch --array=0-31 myarray.sh
```

sjobs shows:

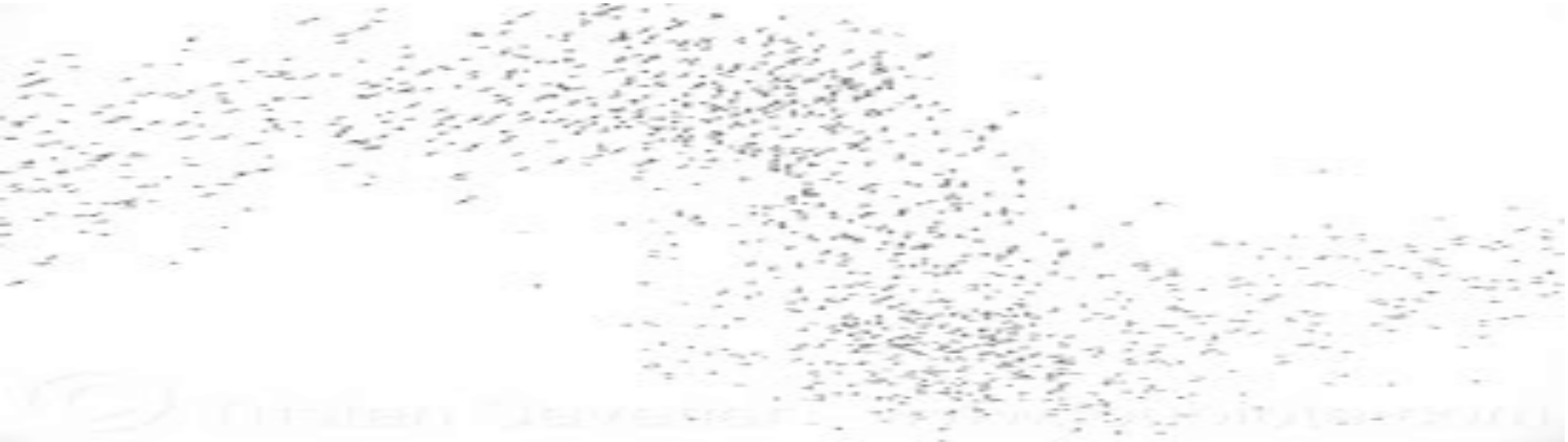
User	JobId	JobName	Part	St	Reason	Runtime	Walltime	Nodes	CPUs	Memory
susanc	49493566_[0-31]	myarray.sh	norm	PD	---	0:00	2:00:00	1	32	2GB/cpu

Limitations of Job Arrays

- Input files are usually not named 0,1,2....
- Likewise, you probably don't want your output files/dirs called 0,1,2,....
- Careful scripting required
- Most batch systems have limitations on array index, # tasks etc.

Swarm

(A flexible & convenient way to submit job arrays)



Submitting swarms of jobs with *swarm*

```
#  
# this file is cmdfile  
#  
myprog -param a < infile-a > outfile-a  
myprog -param b < infile-b > outfile-b  
myprog -param c < infile-c > outfile-c  
myprog -param d < infile-d > outfile-d  
myprog -param e < infile-e > outfile-e  
myprog -param f < infile-f > outfile-f  
myprog -param g < infile-g > outfile-g
```

```
$ swarm -f cmdfile
```

What does swarm do, exactly?

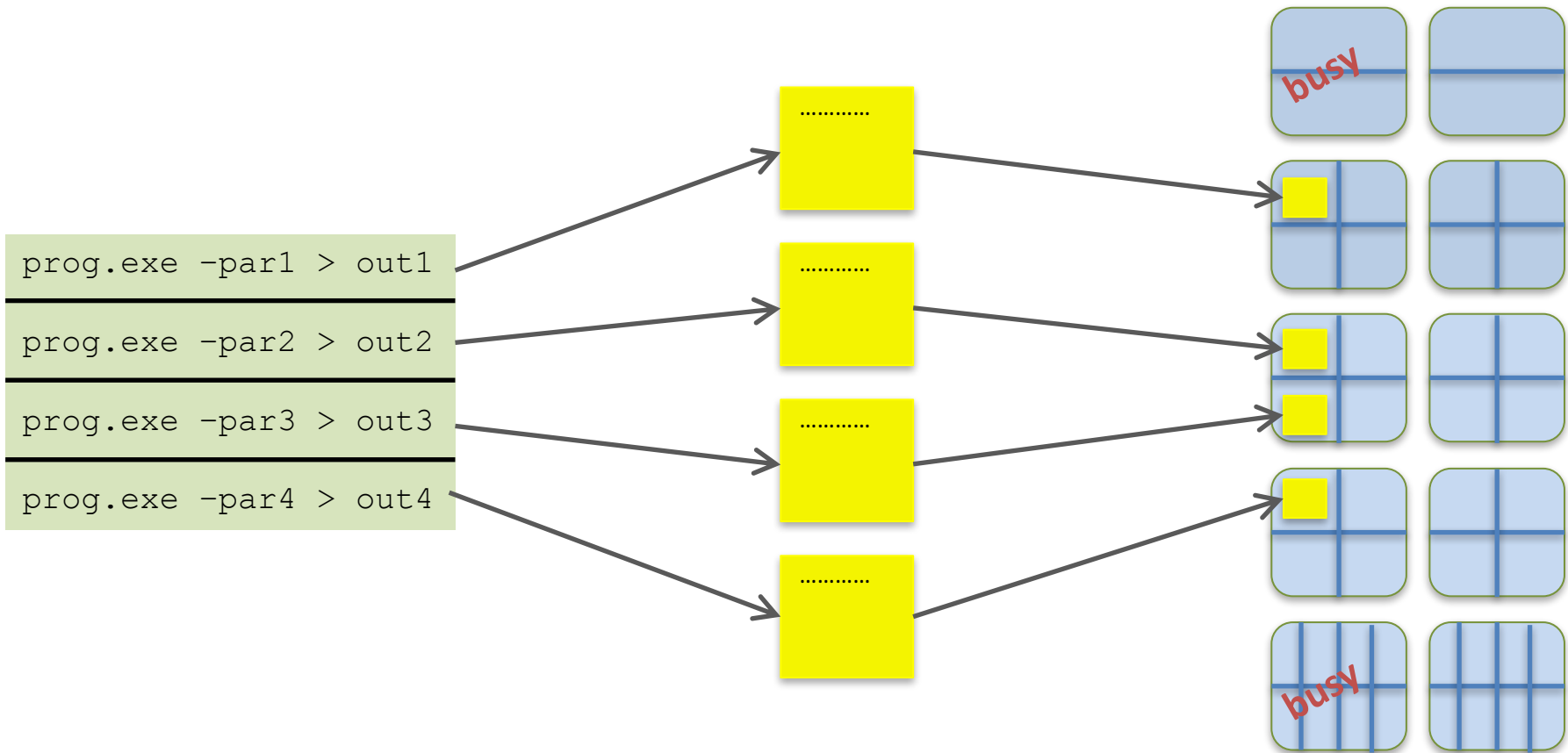
- Creates a subjob for each line in your swarm command file
- Submits all subjobs as a single jobarray to slurm.
- Single jobid regardless of number of commands.
- Subjobs are identified by jobid_arraytaskid, e.g. 12345_1, 12345_2, 12345_4, ...

How swarm works

Swarm command file

Batch scripts

Nodes



4 commands (lines) in swarm command file
1 batch job with 4 subjobs

Swarm of single-threaded processes

Required		Command	Allocated	
CPU	Memory		CPU	Memory
1 per process	< 1.5 GB per process	<code>swarm -f swarmfile</code>	2 per process	1.5 GB per process
1 per process	G GB (G > 1.5)	<code>swarm -g G -f swarmfile</code>	2 per process	G GB per process
1 per process	G GB (G > 1.5)	<code>swarm -g G -p 2 -f swarmfile</code>	1 per process	G GB per process

Swarm of multi-threaded processes

Required		Command	Allocated	
CPU	Memory		CPU	Memory
T threads on T CPUs per process	G GB per process	<code>swarm -t T -g G -f swarmfile</code>	T per process	G GB per process

Default memory allocation: $T \cdot 1.5$ GB per process

Swarm of multi-threaded processes

Use `$SLURM_CPUS_PER_TASK` inside batch script

```
mapsplice.py -1 seq1a.fq -2 seq1b.fq -p $SLURM_CPUS_PER_TASK  
mapsplice.py -1 seq2a.fq -2 seq2b.fq -p $SLURM_CPUS_PER_TASK  
mapsplice.py -1 seq3a.fq -2 seq3b.fq -p $SLURM_CPUS_PER_TASK  
etc
```

Swarm of auto-threading processes

`swarm -t auto`

-> gives you 32 CPUs, possibly overloaded

swarms

- Single “job array” created (one job id, n subjobs)
- The number of subjobs = number of commands
- *swarm* expects bash syntax
- Create complex/long swarm command files with scripts

Swarms: complex commands

Example:

```
cd /data/$USER/mydir1;    somecommand ; some_other_command  
cd /data/$USER/mydir2;    somecommand ; some_other_command  
cd /data/$USER/mydir3;    somecommand ; some_other_command
```

Use semi-colons (;) to separate multiple commands on one line

Can use continuation character (\) to make long lines readable

Each line (set of commands) should be completely independent

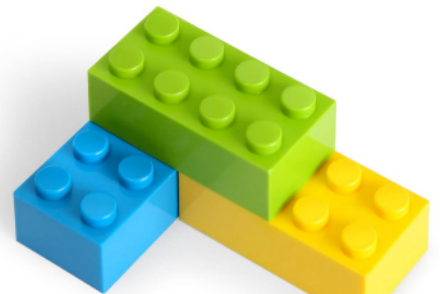
Using modules in swarm scripts

Method 1:

```
bimg -verbose 7 -truncate 0,100 input1.img output1.img  
bimg -verbose 7 -truncate 0,100 input2.img output2.img  
bimg -verbose 7 -truncate 0,100 input3.img output3.img
```

Submit with:

```
swarm -f swarmfile --module bsoft/1.8.2
```



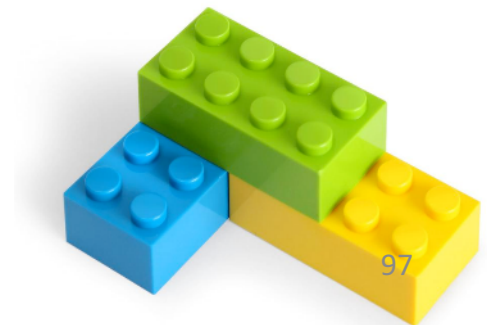
Using modules in swarm scripts

Method 2:

```
module load bsoft/1.8.2; \  
  bimg -verbose 7 -truncate 0,100 input1.img output1.img  
module load bsoft/1.8.2; \  
  bimg -verbose 7 -truncate 0,100 input2.img output2.img  
module load bsoft/1.8.2; \  
  bimg -verbose 7 -truncate 0,100 input3.img output3.img
```

Submit with:

swarm -f swarmfile



Deleting a Swarm

To delete this swarm...

3479375_0	norm	swarm	joeu	R 2-04:15:47	1	cn0175
3479375_5	norm	swarm	joeu	R 2-04:15:47	1	cn0297
3479375_6	norm	swarm	joeu	R 2-04:15:47	1	cn0297
3479375_9	norm	swarm	joeu	R 2-04:15:47	1	cn0304
3479375_10	norm	swarm	joeu	R 2-04:15:47	1	cn0304
3479375_11	norm	swarm	joeu	R 2-04:15:47	1	cn0257
3479375_12	norm	swarm	joeu	R 2-04:15:47	1	cn0257
3479375_13	norm	swarm	joeu	R 2-04:15:47	1	cn0257

```
$ scancel 3479375
```

Hands-on (swarm)

```
cd /data/$USER/hpc-classes/biowulf/swarm
```

- Use 'more' to examine the file blat.swarm
- 'wc -l blat.swarm' will give you the # of commands

```
swarm [--partition student] -f blat.swarm --module blat
```

- How many jobs were created?
- Use 's jobs' to watch your jobs.
- Use 'ls' and 'more' to examine the output after the swarm is done.

Swarm options

```
$ swarm --help
```

```
Usage: swarm [swarm options] [qsub options]
```

```
Usage: swarm [swarm options] [sbatch options]
```

-f, --file [file]	name of file with list of command lines to execute, with a single command line per subjob
-g, --gb-per-process [float]	gb per process (can be fractions of GB, e.g. 3.5)
-t, --threads-per-process [int]/"auto"	threads per process (can be an integer or the word auto). This option is only valid for multi-threaded swarms (-p 1).
-p, --processes-per-subjob [int]	processes per subjob (default = 1). This option is only valid for single-threaded swarms (-t 1).
-b, --bundle [int]	bundle more than one command line per subjob and run sequentially
--usecsh	use tcsh as the shell instead of bash
-m, --module	provide a list of environment modules to load prior to execution (comma delimited)

Swarm options

\$ **swarm --help [contd]**

--no-comment don't ignore text following comment character #
-c, --comment-char [chr] use something other than # as the comment character

--logdir directory to which .o and .e files are to be written
(default is current working directory)

--maxrunning limit the number of simultaenously running subjobs

sbatch options:

-J, --job-name [str] set the name of the job
--dependency [str] set up dependency (i.e. run swarm before or after)
--time [str] change the walltime for each subjob (default is 04:00:00, or 4 hours)

-L, --licenses [str] obtain software licenses (e.g. --licenses=matlab)
--partition [str] change the partition (default is norm)
--gres [str] set generic resources for swarm
--qos [str] set quality of service for swarm

Other sbatch options

--sbatch [string] add sbatch-specific options to swarm. These options will be added last, which means that swarm options for allocation of cpus and memory take precedence.

Swarm bundles

For large swarms or swarms with short-lived threads (under 5-10 minutes)

```
swarm -b 20 -f commandfile
```

creates a “bundle” (queue) of 20 processes *per core*

- Example: 8000 lines (8000 processes), -b 20 => $8000/20 = 400$ jobs
- Swarm will ‘auto-bundle’ for command files with > 1,000 lines

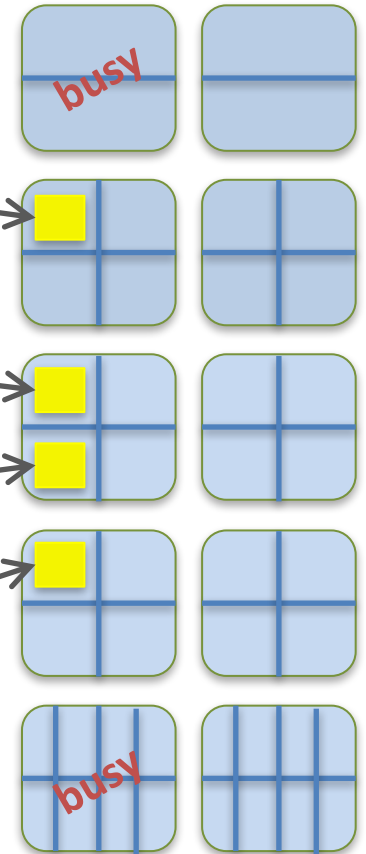
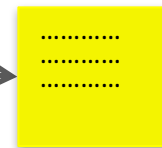
How swarm bundling works

Swarm command file

Batch scripts

Nodes

```
prog.exe -par1 > out1  
prog.exe -par2 > out2  
prog.exe -par3 > out3  
-----  
prog.exe -par4 > out4  
prog.exe -par5 > out5  
prog.exe -par6 > out6  
-----  
prog.exe -par7 > out7  
prog.exe -par8 > out8  
prog.exe -par9 > out9  
-----  
prog.exe -para > outa  
prog.exe -parb > outb  
prog.exe -parc > outc
```



12 commands (lines) in swarm command file
1 batch job with 4 subjobs
Each subjob runs 3 commands sequentially

Quiz

What's the difference between:

`swarm -p 2 -f swarmfile`

`swarm -b 2 -f swarmfile`

Answer

`swarm -p 2 -f swarmfile`

2 processes run simultaneously per core (1 process per CPU)

`swarm -b 2 -f swarmfile`

2 processes run sequentially per core (may still leave 1 CPU idle)

Hands-on (swarm bundling)

```
cd /data/$USER/hpc-classes/biowulf/swarm  
swarm [--partition=student] -b 4 -g 4 -f blat.swarm \  
--module blat
```

How many subjobs were produced?

sbatch vs swarm

sbatch	swarm
Batch system submission command	Swarm uses sbatch in the background
Write your own batch scripts	Write a file containing a list of commands
Can be inconvenient for large numbers of similar jobs	Very convenient for large numbers of similar jobs
Easier to debug a failed job	Can be difficult to debug a failed subjob

Hands-on (swarm vs sbatch)

```
cd /data/$USER/hpc-classes/biowulf/R
```

Compare the files R.bat and swarm.cmd

How would you submit each one?

More on Biowulf/Slurm Resources

Partitions (Queues)

```
$ batchlim
```

```
Max jobs per user: 4000
```

```
Max array size: 1001
```

Partition	MaxCPUsPerUser	DefWalltime	MaxWalltime

norm	6144	02:00:00	10-00:00:00
multinode	6272	08:00:00	10-00:00:00
turbo qos	12544		08:00:00
interactive	64	08:00:00	1-12:00:00 (3 simultaneous jobs)
quick	6144	02:00:00	04:00:00
largemem	512	04:00:00	10-00:00:00
gpu	224	02:00:00	10-00:00:00 (16 GPUs per user)
unlimited	128	UNLIMITED	UNLIMITED
student	32	02:00:00	04:00:00 (2 GPUs per user)
ccr	3072	04:00:00	10-00:00:00
ccrgpu	448	04:00:00	10-00:00:00 (32 GPUs per user)
ccrclin	224	04:00:00	10-00:00:00
ccrlcb	1024	04:00:00	10-00:00:00
ccrlcbgpu		04:00:00	10-00:00:00
niddk	1024	04:00:00	10-00:00:00
nimh	1024	04:00:00	10-00:00:00

Show available compute resources

% **freen**

.....Per-Node Resources.....

Partition	FreeNds	FreeCPUs	Cores	CPUs	Mem	Disk	Features
norm*	0/388	6672/21728	28	56	248g	400g	cpu56,core28,g256,ssd400,x2695,ibfdr
norm*	0/310	1642/9920	16	32	123g	800g	cpu32,core16,g128,ssd800,x2650,10g
norm*	274/282	4406/4512	8	16	21g	200g	cpu16,core8,g24,sata200,x5550,1g
unlimited	0/3	22/168	28	56	248g	400g	cpu56,core28,g256,ssd400,x2695,ibfdr
unlimited	13/13	416/416	16	32	123g	800g	cpu32,core16,g128,ssd800,x2650,10g
niddk	52/86	1822/2752	16	32	123g	800g	cpu32,core16,g128,ssd800,x2650,10g,niddk
multinode	37/487	5686/27272	28	56	248g	400g	cpu56,core28,g256,ssd400,x2695,ibfdr
multinode	3/186	486/5952	16	32	60g	800g	cpu32,core16,g64,ssd800,x2650,ibfdr
gpu	15/20	510/640	16	32	123g	800g	cpu32,core16,g128,ssd800,x2650,gpuk20x
gpu	3/3	96/96	16	32	123g	800g	cpu32,core16,g128,ssd800,x2650,gpuk20x,acemd
gpu	1/1	32/32	16	32	123g	800g	cpu32,core16,g128,ssd800,x2650,gpuk20x,acemd,desm
largemem	2/3	148/192	32	64	1007g	800g	cpu64,core32,g1024,ssd800,x4620,10g
nimh	62/63	1992/2016	16	32	123g	800g	cpu32,core16,g128,ssd800,x2650,10g,nimh
ccr	1/93	1518/2976	16	32	123g	800g	cpu32,core16,g128,ssd800,x2650,10g,ccr
ccr	39/122	6432/6832	28	56	248g	400g	cpu56,core28,g256,ssd400,x2695,ibfdr,ccr
ccrlcb	34/34	1904/1904	28	56	248g	400g	cpu56,core28,g256,ssd400,x2695,ibfdr,ccrlcb
ccrclin	4/4	224/224	28	56	248g	400g	cpu56,core28,g256,ssd400,x2695,ibfdr,ccrclin
ccrgpu	12/16	880/896	28	56	248g	400g	cpu56,core28,g256,ssd400,x2695,ibfdr,gpuk80,ccr
ccrlcbgpu	5/5	280/280	28	56	248g	400g	cpu56,core28,g256,ssd400,x2695,ibfdr,gpuk80,ccrlcb
quick	1/93	1518/2976	16	32	123g	800g	cpu32,core16,g128,ssd800,x2650,10g,ccr
quick	39/122	6432/6832	28	56	248g	400g	cpu56,core28,g256,ssd400,x2695,ibfdr,ccr
quick	12/16	880/896	28	56	248g	400g	cpu56,core28,g256,ssd400,x2695,ibfdr,gpuk80,ccr
quick	52/86	1822/2752	16	32	123g	800g	cpu32,core16,g128,ssd800,x2650,10g,niddk
quick	1/63	32/2016	16	32	29g	400g	cpu32,core16,g32,sata400,x2600,ibqdr
quick	0/66	0/2112	16	32	60g	400g	cpu32,core16,g64,sata400,x2600,ibqdr
quick	62/63	1992/2016	16	32	123g	800g	cpu32,core16,g128,ssd800,x2650,10g,nimh
quick	61/61	1952/1952	16	32	123g	400g	cpu32,core16,g128,sata400,x2600,1g
quick	64/64	2048/2048	16	32	60g	400g	cpu32,core16,g64,sata400,x2600,1g
quick	345/350	8380/8400	12	24	21g	100g	cpu24,core12,g24,sata100,x5660,1g

Submitting Jobs to Partitions

```
sbatch --partition=quick jobscript
```

```
swarm --partition=quick swarmfile
```

Submitting Jobs to Partitions

You can submit to 2 partitions:

```
sbatch --partition=ccr,norm jobscript
```

and the job will run on whichever partition has the requested resources.

CPU limits in partitions are cumulative, example for NIDDK user:

niddk (1024 cpus) + norm (6144 cpus) + quick (6144 cpus) = 13312 cpus

Walltime

The max time you expect your job to run

- All jobs have a walltime limit
- Default 2 hr walltime in norm (default) partition
- Specify walltime with

```
--time=d-hh:mm:ss
```

e.g. --time=12:00:00 (12 hrs)
 --time=36:00:00 (36 hrs)
 --time=2-12:00:00 (2 days, 12 hrs)



Increasing or decreasing walltime

```
newwall --jobid <job_id> --time <new_time_spec>
```

Use Slurm's email alerts to track job walltimes and runtimes

```
sbatch --mail-type=BEGIN,TIME_LIMIT_90,END batch_script.sh
```

TIME_LIMIT_50	Job reached 50% of its time limit
TIME_LIMIT_80	Job reached 80% of its time limit
TIME_LIMIT_90	Job reached 90% of its time limit
TIME_LIMIT	Job reached its time limit



Choosing a reasonably accurate walltime is important



Swarm & bundled walltimes

```
swarm -f swarmfile --time 2:00:00
```



Time for 1 command in swarmfile

```
swarm -f swarmfile --time 2:00:00 -b 20
```



Swarm will multiply 2 hrs * 20 and set each job to have a walltime of 40 hrs.
(Error if > partition time limit)

“Unlimited” Partition

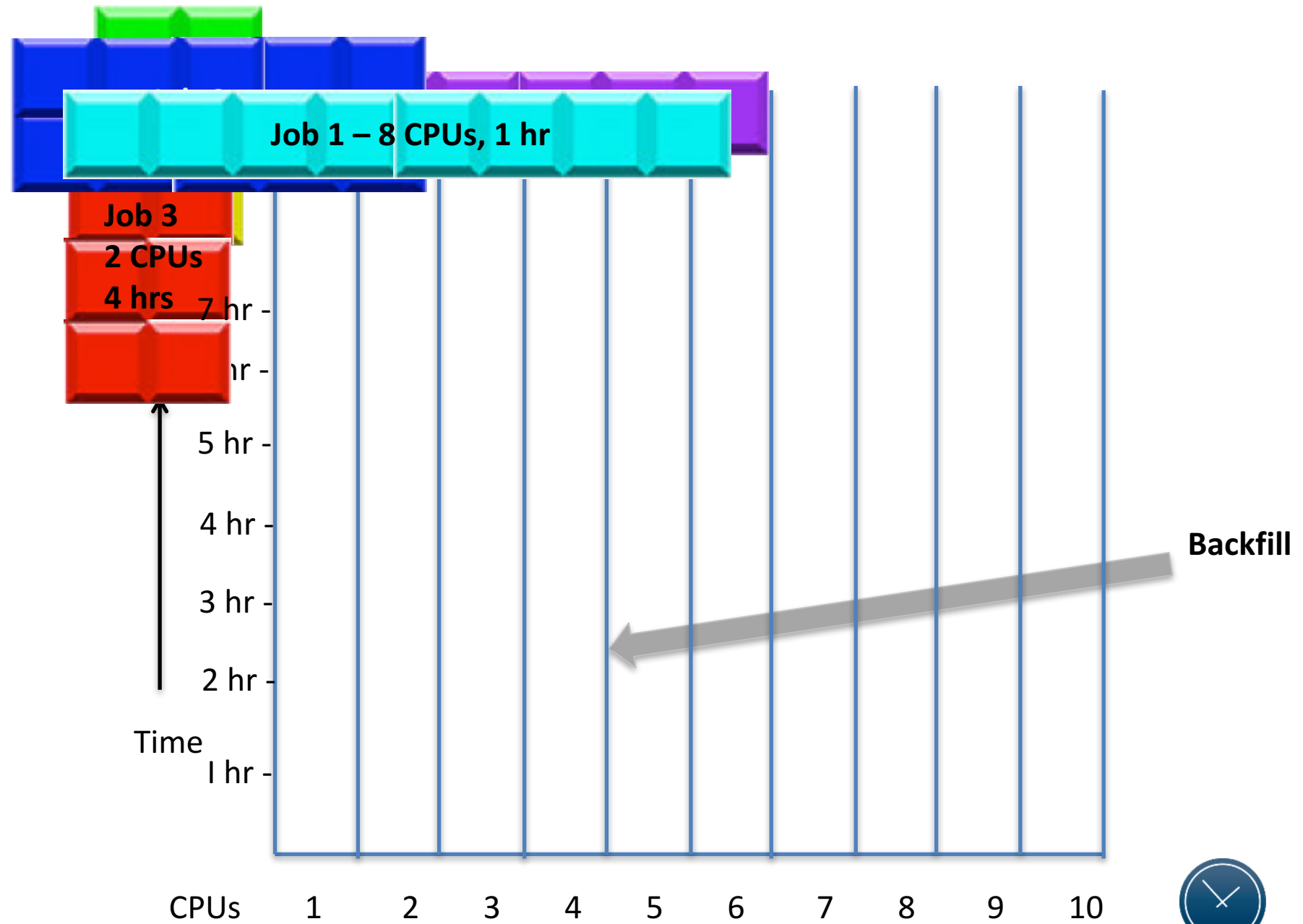
```
sbatch --partition=unlimited ...
```

- No walltime limit
- Very limited number of nodes available

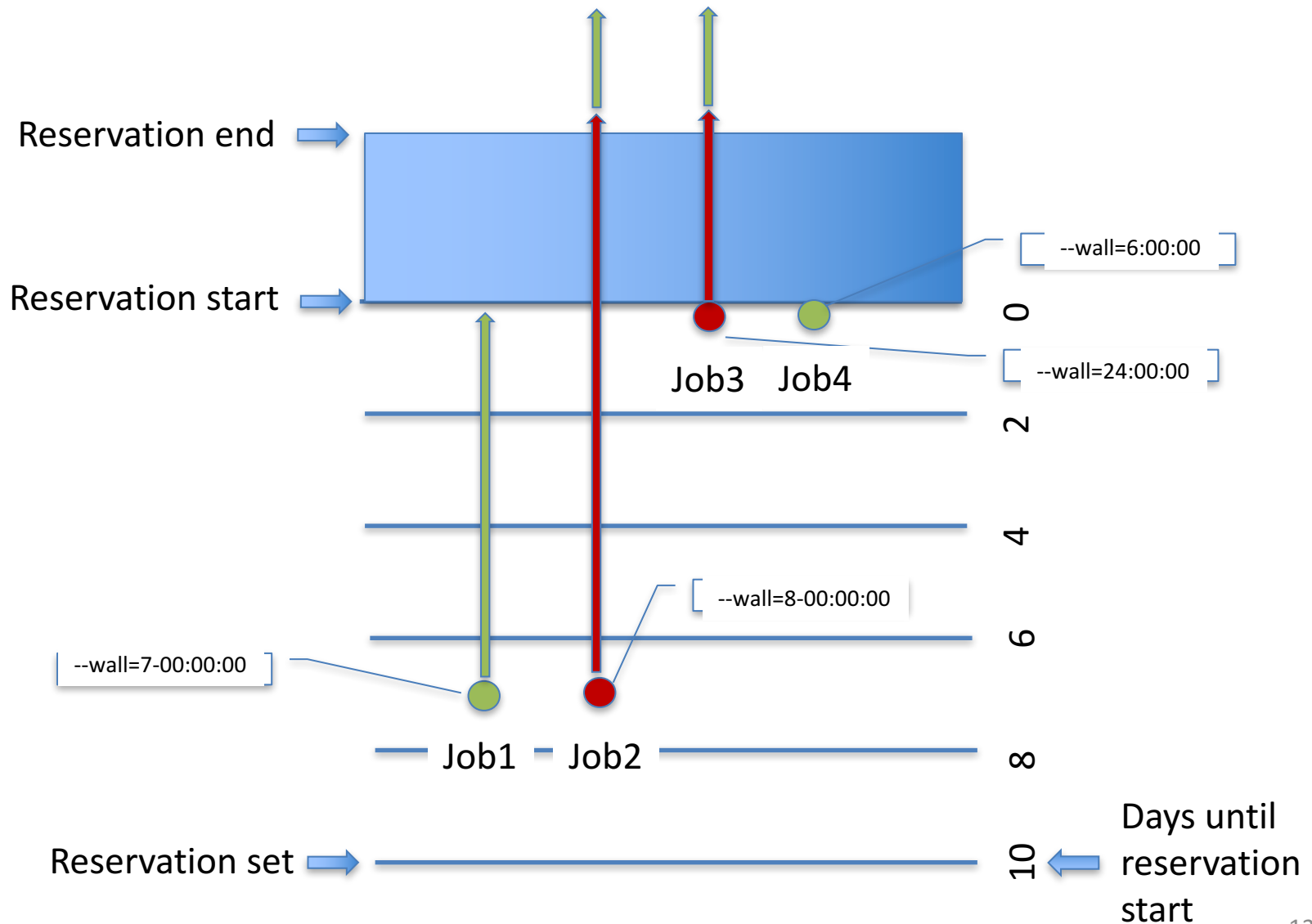
How Jobs are Scheduled

- Within each partition scheduled by priority
- Priority primarily based on Fairshare
- Fairshare is the number of cpu-minutes used recently (half life of 7 days)
- `sprio` command reports priorities of pending (queued) jobs
- Backfill job scheduling for parallel jobs

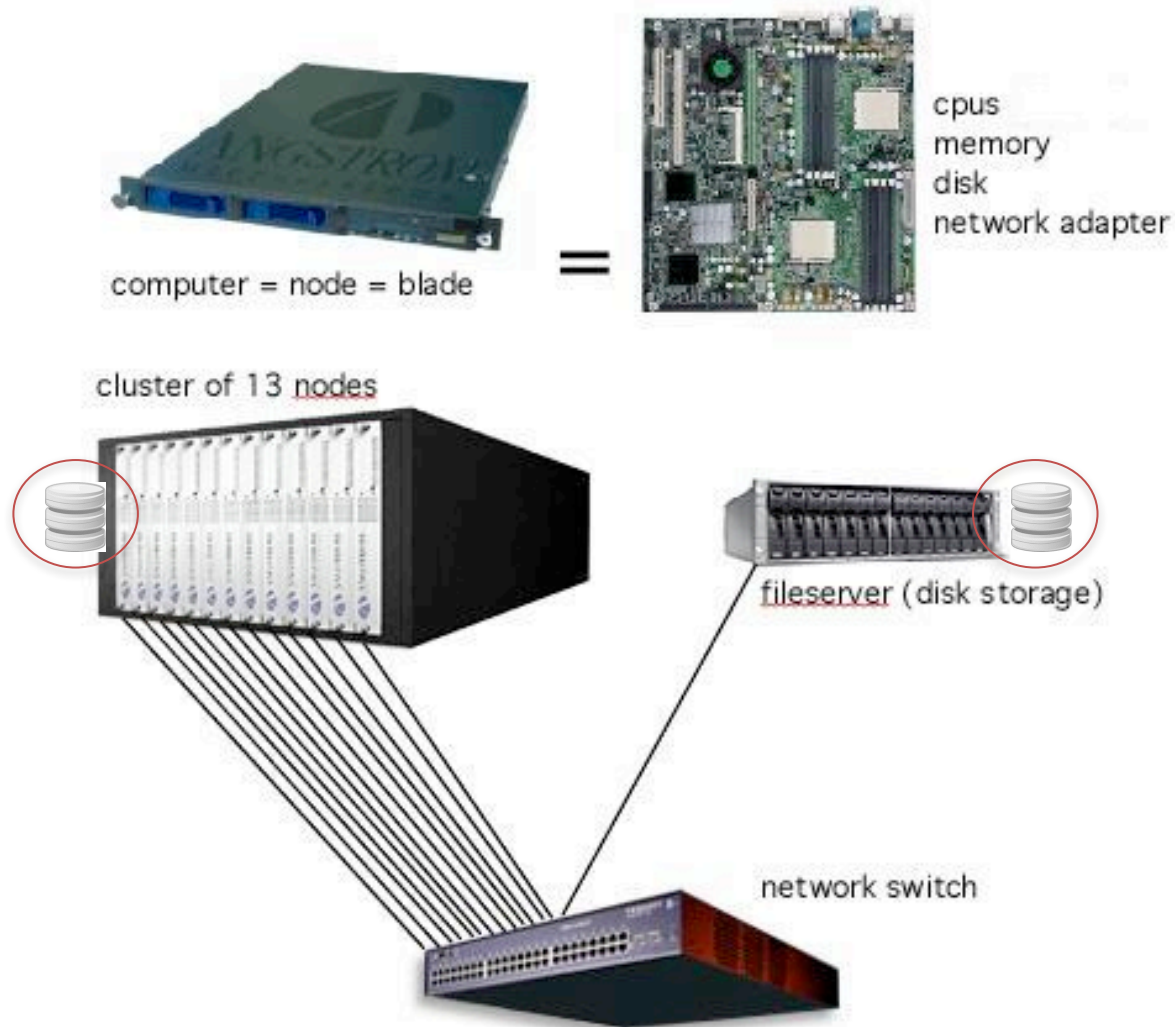
Why accurate walltimes are important (Backfill scheduling)



Another reason why accurate walltimes are important: Maintenance window reservations



Local disk on node



Local disk on node

- Local disk = /lscratch
- (/scratch is central shared space)
- 'freen' will report disk size as a Feature (eg, ssd800 = 800GB SSD disk)
- Jobs which need to use /lscratch must explicitly allocate space:

```
sbatch --gres=lscratch:500 ...
```

↑
GB

- Read/write to /lscratch/\$SLURM_JOBID (/lscratch directory not writable)
- /lscratch/\$SLURM_JOBID deleted at end of job

More Best Practices for Storage

BAD	GOOD
100 jobs all reading the same 50GB file over and over from /data/\$USER/	Use /lscratch instead, copy the file there, and have each job access the file on local disk.
100 jobs all writing and deleting large numbers of small temporary files.	Use /lscratch instead, have all tmp files written to local disk.

Hands-on (lscratch)

```
sinteractive --gres=lscratch:2  
      (Requesting 2 GB of local scratch)
```

Once you are logged into a node:

```
cd /lscratch/$SLURM_JOBID; ls -l  
cp -r /data/$USER/hpc-classes/biowulf/freebayes/ .  
module load freebayes  
cd freebayes  
freebayes -f genome.fasta input.bam  
exit
```

Question: once you exit the job, can you access the files on /lscratch on the node?
What happens if you try to write to /lscratch, instead of /lscratch/\$SLURM_JOBID?

Graphics Processing Units



48 nVIDIA K20x

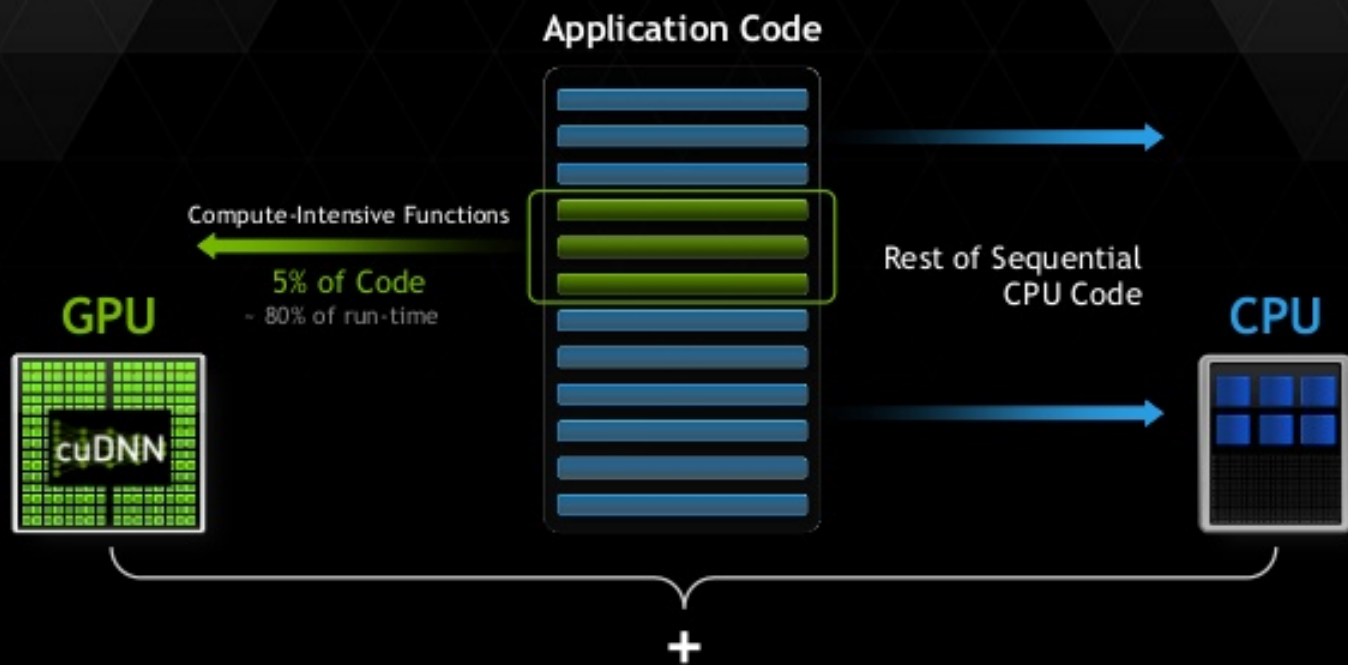
48 nVIDIA K80 (CCR buy-in)

288 nVIDIA K80

192 nVIDIA P100 (Phase 4)

32 nVIDIA V100

HOW GPU ACCELERATION WORKS





GPU applications

Molecular Dynamics

NAMD
Gromacs
Amber
Acemd
Charmm
Q-Chem

Image Analysis

Relion

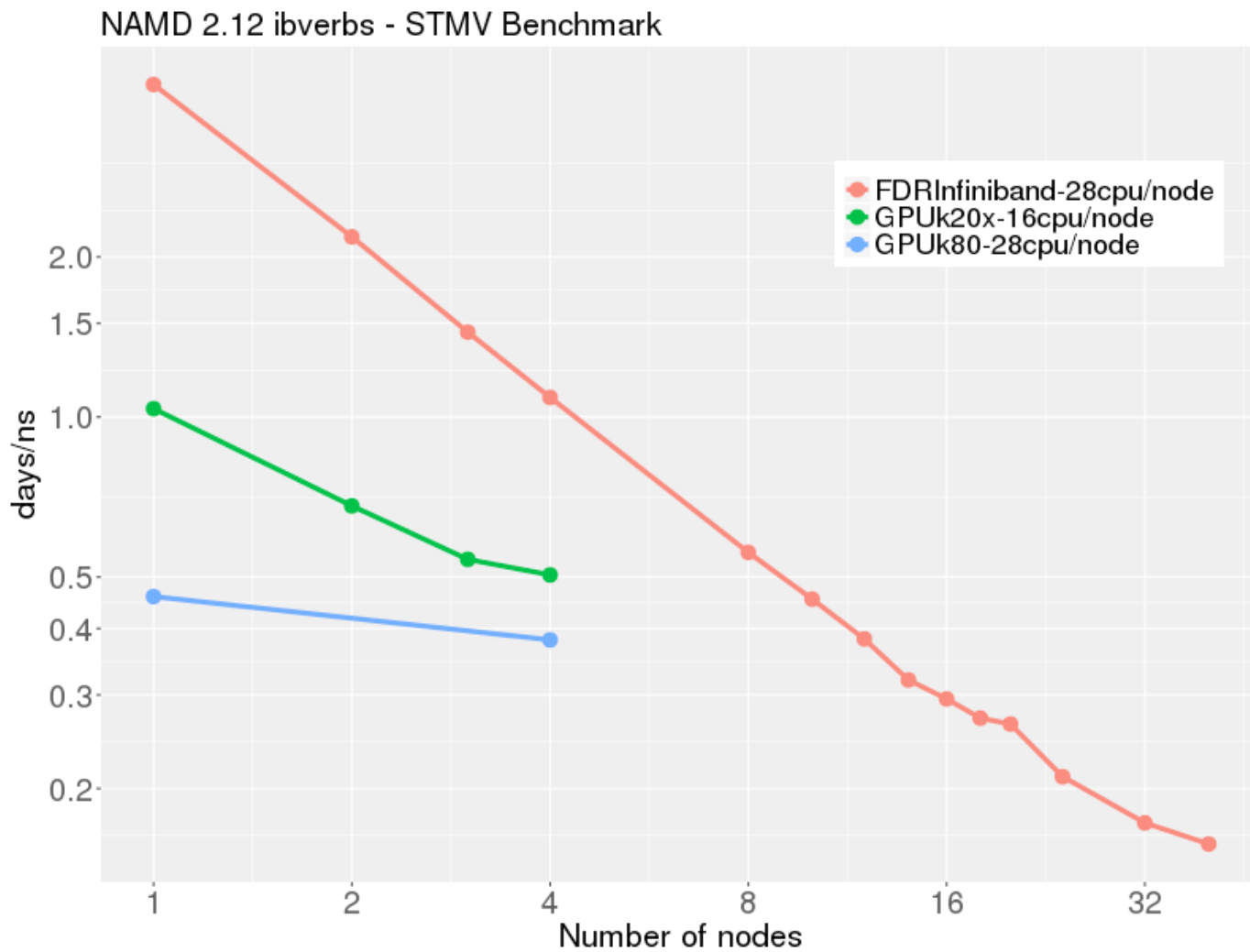
Deep Learning

Caffe
TensorFlow
Digits

And Others...

SIFT
BEAST
Matlab

CPUs vs GPU



GPU Nodes on Biowulf

#nodes	GPU type	#GPUs/ node	#CPUs/ node	#CPUs/GPU slurm binding
24	K20x	2	16	8
72	K80	4	56	14
24 (NCI/CCR)	K80	4	56	14
48	P100	4	56	14
8	V100	4	56	14



Allocating GPUs

```
sbatch --partition=gpu --gres=gpu:k80:1 jobscript
```



```
sbatch --partition=gpu --gres=gpu:k80:2 jobscript
```

Resource Name	Resource Type	Count Required even if 1
•k20x		(Max of 2 K20x
•k80		4 K80)

(for student accounts, replace --partition=gpu with --partition=student)

Large Memory Jobs (>256 GB)

```
sbatch --mem=768g --partition=largemem ...
```

- 28 large memory nodes
- Use large memory nodes for their memory, not for their (144) cores  

1.5TB & 3TB nodes – can anyone here make use of these?

Is there a need for a 144-thread single-node job?

Licensed Software

- Only compiled Matlab code can be run as batch jobs
- All other Matlab jobs must be run interactively
- Use the 'licenses' command to see current status of licenses

Example of batch job using license

```
sbatch --partition gpu --gres=gpu:k80:1 --license=acemd jobscript
```

Monitoring Batch Jobs



- Things to monitor
 - Core (CPU) utilization
 - Memory
- Tools for monitoring
 - `sjobs`, `squeue` (show status of jobs)
 - `jobload` (show CPU & mem usage)
 - `jobhist` (show utilization after job ends)
 - Dashboard <https://hpc.nih.gov/dashboard>
 - (https://hpc.nih.gov/nih/student_dashboard/)
- *Users* (you) are responsible for monitoring their jobs

Monitoring – show all jobs



% squeue

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
22340	multinode	stmv-204	susanc	PD	0:00	128	(Dependency)
22353	multinode	meme_sho	susanc	PD	0:00	128	(Dependency)
22339	multinode	stmv-102	susanc	PD	0:00	64	(Dependency)
22352	multinode	meme_sho	susanc	PD	0:00	64	(Dependency)

% sjobs

User	JobId	JobName	Part	St	Runtime	Nodes	CPUs	Mem	Dependency	Features	Nodelist
susanc	22344	meme_short	norm	R	22:30	1	32	1.0GB/cpu		(null)	cn0414
susanc	22335	stmv-64	multinode	R	0:54	4	128	1.0GB/cpu		(null)	cn[0413-0415]

Monitoring – jobload



```
% jobload -u username
```

JOBID	TIME	NODES	CPUS	THREADS	LOAD	MEMORY
	Elapsed / Wall			Alloc	Active	Used / Alloc
35268348	18:46:09 /	1-06:00:00	cn1278	56	28	50% 8.6 / 56.0 GB
	18:46:09 /	1-06:00:00	cn1279	56	28	50% 8.5 / 56.0 GB
	18:46:09 /	1-06:00:00	cn1280	56	28	50% 8.6 / 56.0 GB
	18:46:09 /	1-06:00:00	cn1281	56	28	50% 8.3 / 56.0 GB
	18:46:09 /	1-06:00:00	cn1282	56	28	50% 7.9 / 56.0 GB
	Nodes: 5	CPUs: 280	Load Avg: 50%			
35274295	18:21:51 /	1-06:00:00	cn1297	56	28	50% 9.6 / 56.0 GB
	18:21:51 /	1-06:00:00	cn1298	56	28	50% 9.4 / 56.0 GB
	18:21:51 /	1-06:00:00	cn1299	56	28	50% 9.6 / 56.0 GB
	18:21:51 /	1-06:00:00	cn1300	56	28	50% 9.3 / 56.0 GB
	18:21:51 /	1-06:00:00	cn1301	56	28	50% 8.8 / 56.0 GB
USER SUMMARY						
Jobs: 2	Nodes: 10	CPUs: 560	Load Avg: 50%			

```
% jobload -j 35274295
```

JOBID	TIME	NODES	CPUS	THREADS	LOAD	MEMORY
	Elapsed / Wall			Alloc	Active	Used / Alloc
35274295	18:23:34 /	1-06:00:00	cn1297	56	28	50% 9.6 / 56.0 GB
	18:23:34 /	1-06:00:00	cn1298	56	28	50% 9.5 / 56.0 GB
	18:23:34 /	1-06:00:00	cn1299	56	28	50% 9.6 / 56.0 GB
	18:23:34 /	1-06:00:00	cn1300	56	28	50% 9.3 / 56.0 GB
	18:23:34 /	1-06:00:00	cn1301	56	28	50% 8.8 / 56.0 GB
	Nodes: 5	CPUs: 280	Load Avg: 50%			



Monitoring - jobload

Parallel MPI jobs may show “0%” load – this is an (undesirable) “feature”

```
$ jobload -u abramyanam
```

JOBID	TIME	NODES	CPUS	THREADS	LOAD	MEMORY
	Elapsed / Wall		Alloc	Active		Used / Alloc
35329432	1-13:18:36 / 10-00:00:00	cn1351	56	0	0%	0.0 / 56.0 GB
	1-13:18:36 / 10-00:00:00	cn1352	0	0	0%	0.0 / 0.0 GB
	1-13:18:36 / 10-00:00:00	cn1353	0	0	0%	0.0 / 0.0 GB
	1-13:18:36 / 10-00:00:00	cn1354	0	0	0%	0.0 / 0.0 GB
	1-13:18:36 / 10-00:00:00	cn1355	0	0	0%	0.0 / 0.0 GB
	1-13:18:36 / 10-00:00:00	cn1356	0	0	0%	0.0 / 0.0 GB
	1-13:18:36 / 10-00:00:00	cn1357	0	0	0%	0.0 / 0.0 GB
	1-13:18:36 / 10-00:00:00	cn1358	0	0	0%	0.0 / 0.0 GB
	1-13:18:36 / 10-00:00:00	cn1359	0	0	0%	0.0 / 0.0 GB
	1-13:18:36 / 10-00:00:00	cn1360	0	0	0%	0.0 / 0.0 GB

jobhist



```
[biowulf ~]$ jobhist 22343
```

```
JobId           : 22343
User            : susanc
Submitted       : 20150522 08:59:54
Submission Dir   : /spin1/users/susanc/meme
Submission Command : sbatch --depend=afterany:22342 --partition=multinode --ntasks=2
                  --ntasks-per-core=1  meme_short.slurm
```

Partition	State	AllocNodes	AllocCPUs	Walltime	MemReq	MemUsed	Nodelist
multinode	COMPLETED	1	32	01:43:58	1.0GB/cpu	0.7GB	cn0414

Note: RUNNING jobs will always show MemUsed = 0.0GB

Monitoring - Dashboard

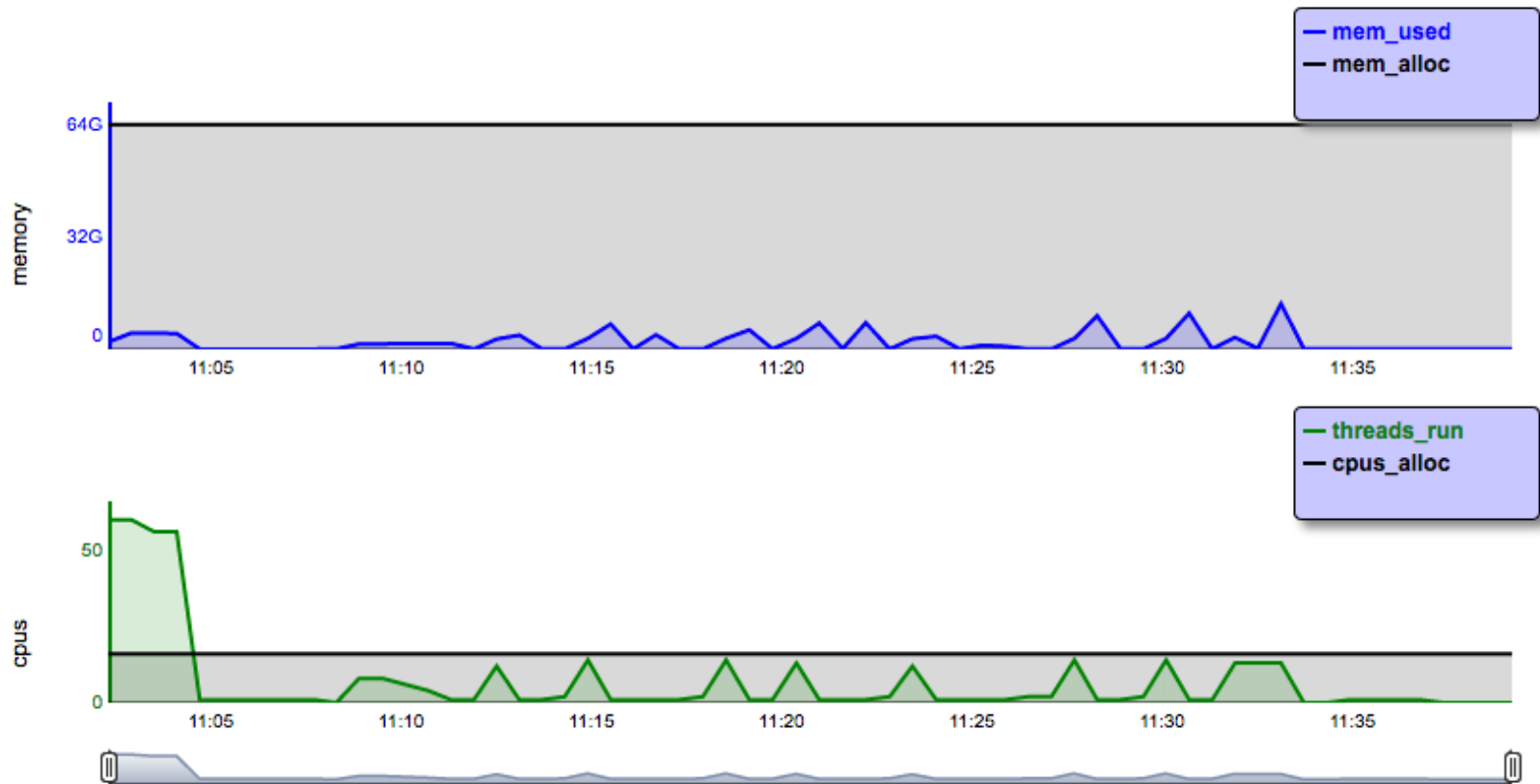


<https://hpc.nih.gov/dashboard>

Student accounts: https://hpc.nih.gov/nih/student_dashboard

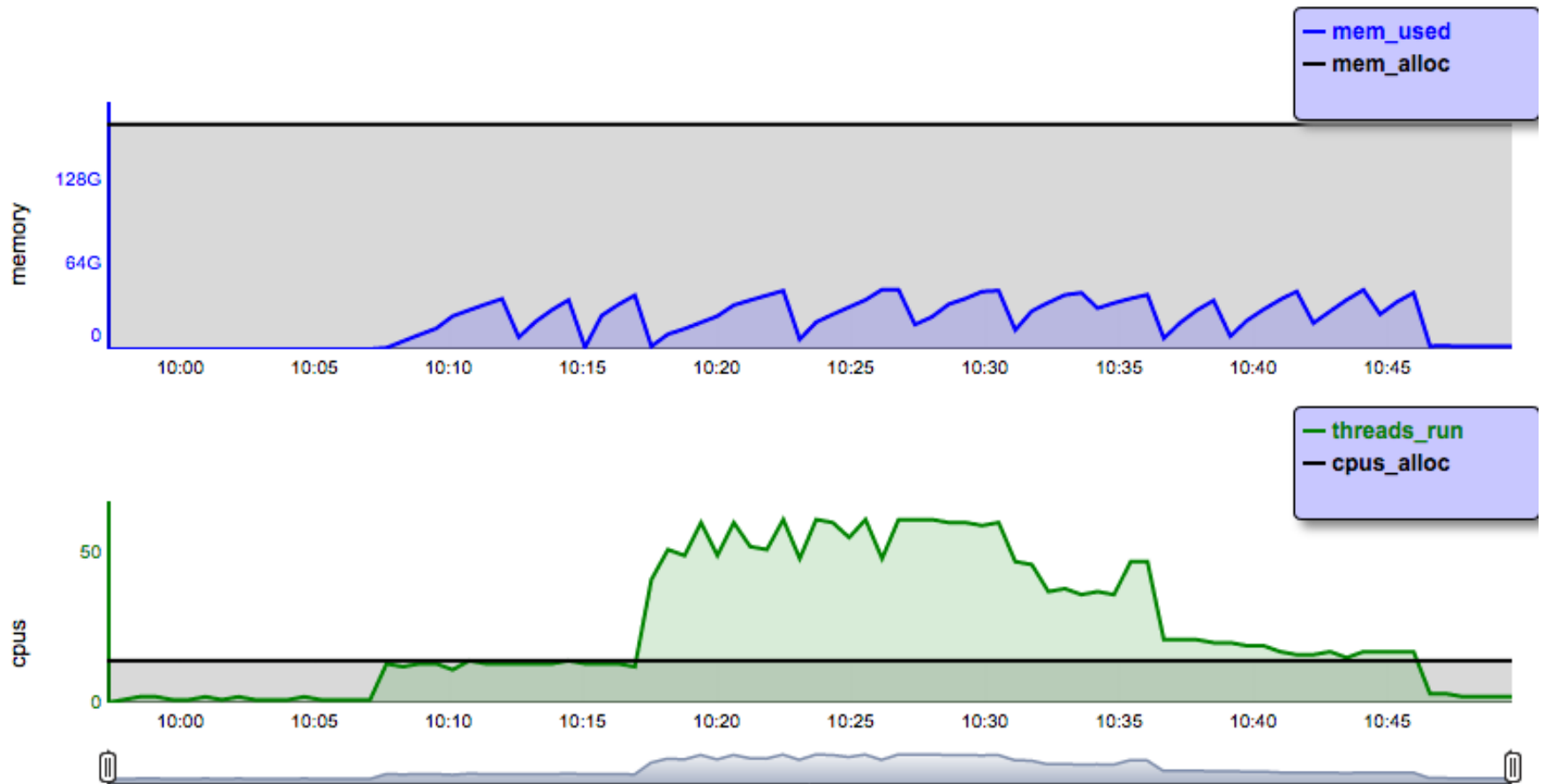
Demo

Dashboard examples

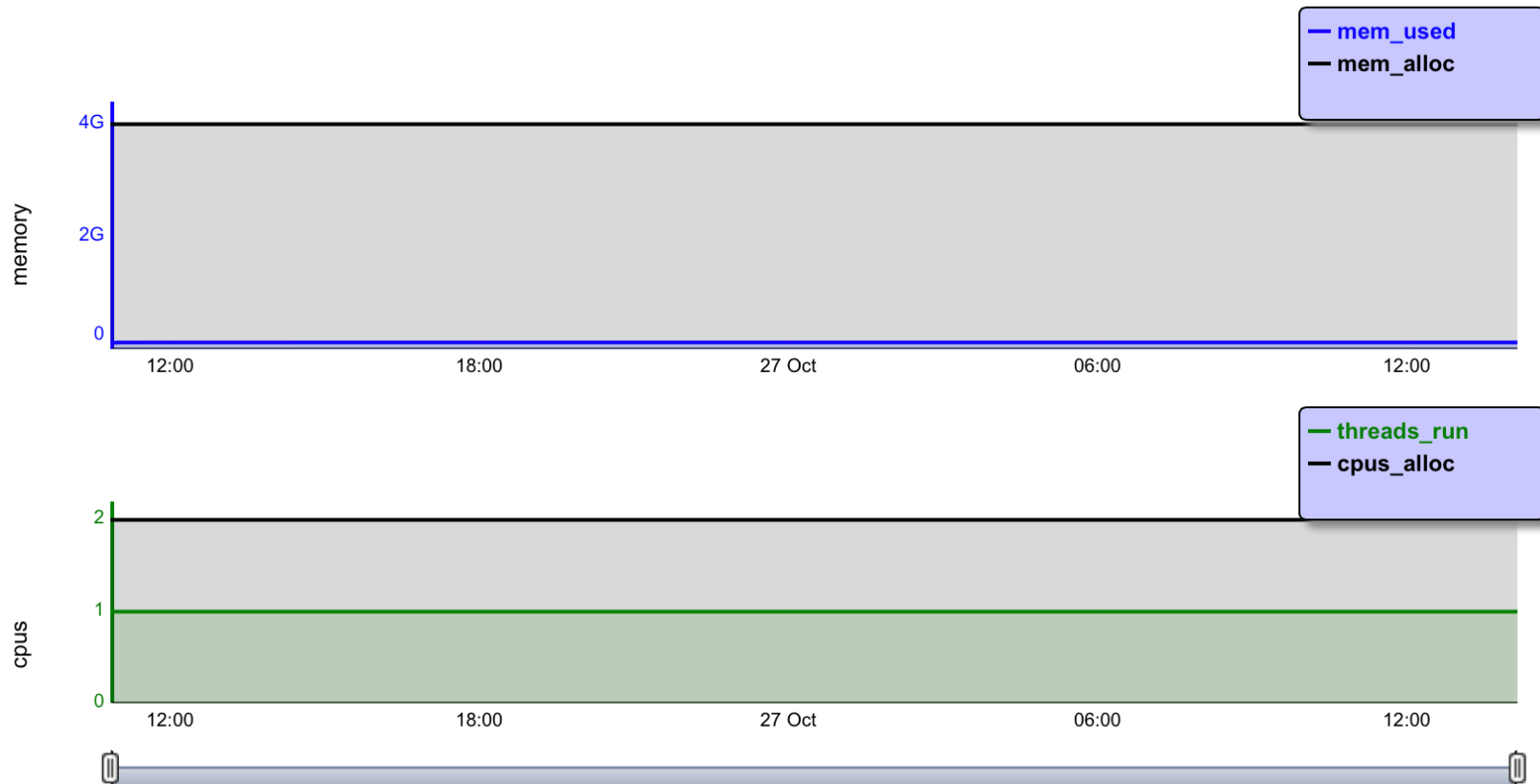


CPUs and memory allocation are fine

Another dashboard example

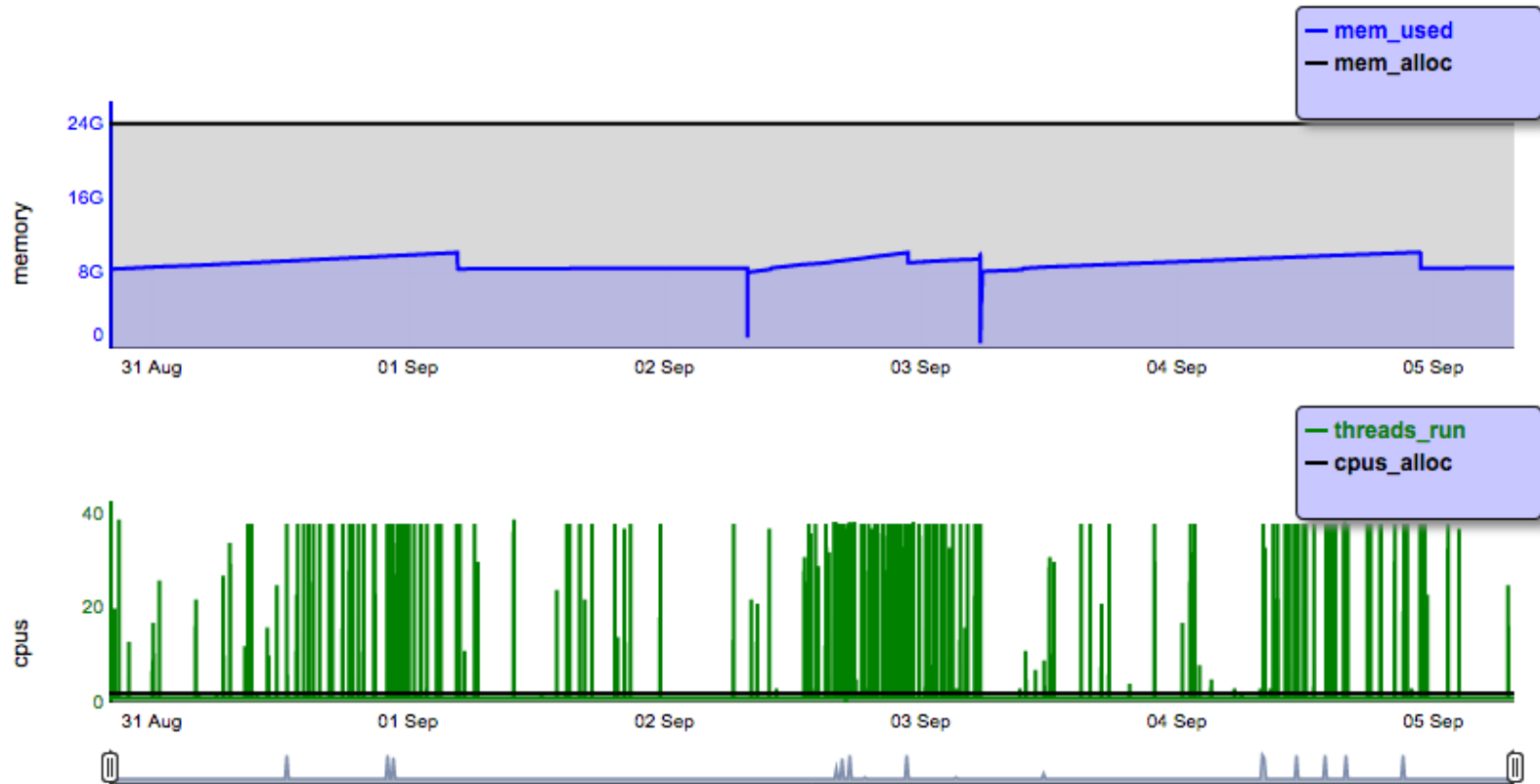


Recommendation: might want to increase CPU allocation to 56

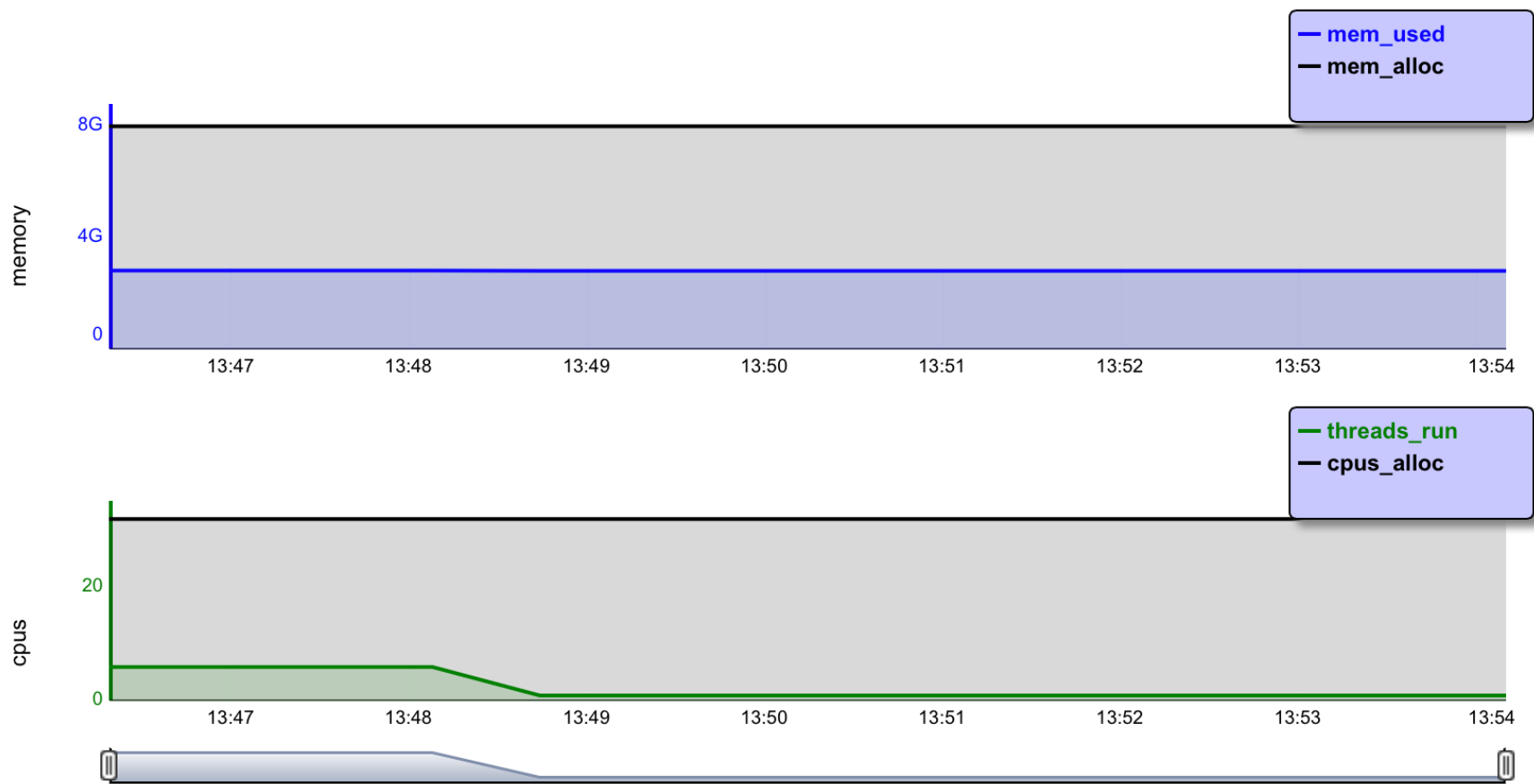


Comment: job is running with default allocations for CPU and memory
Recommendation: if a subjob of a large swarm, try “-p 2”

Another dashboard example

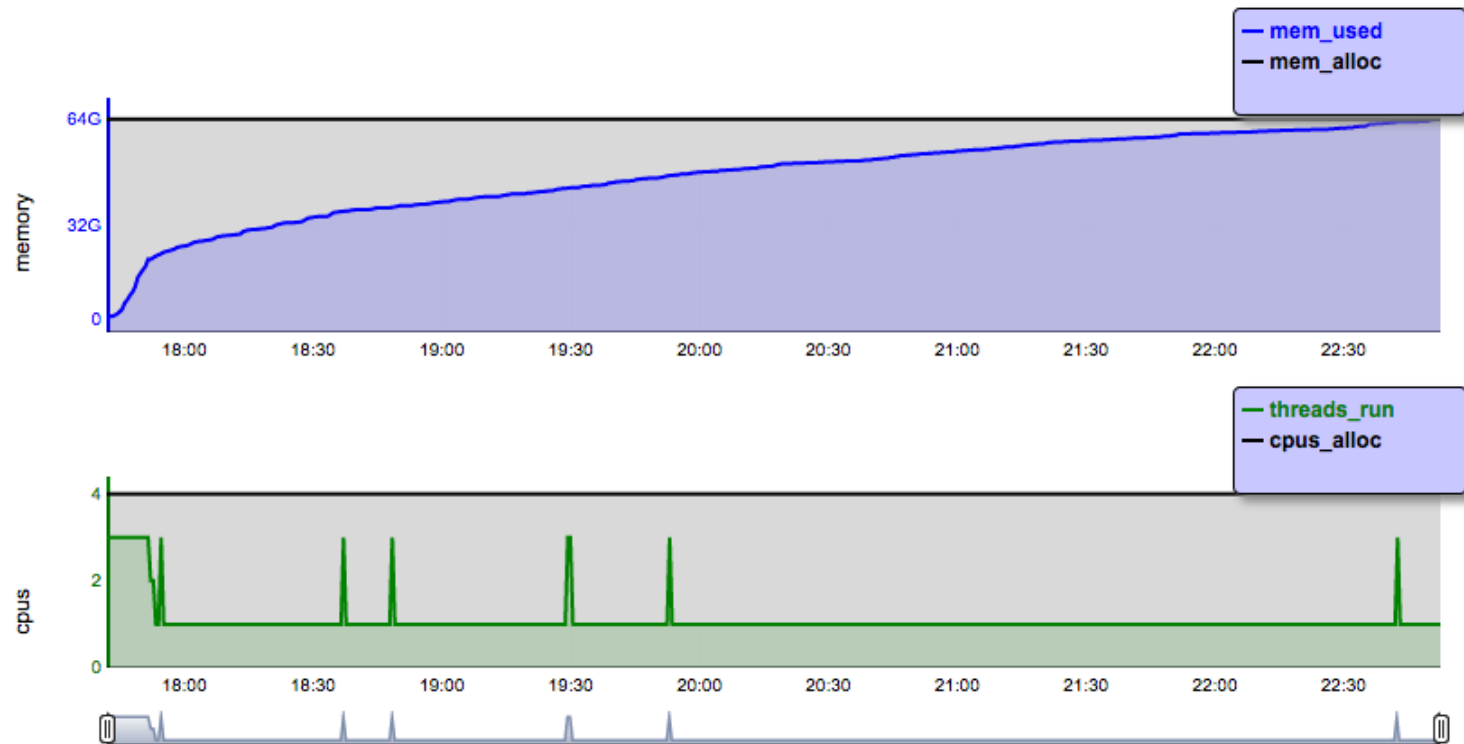


Recommendation: should increase CPU allocation to 40



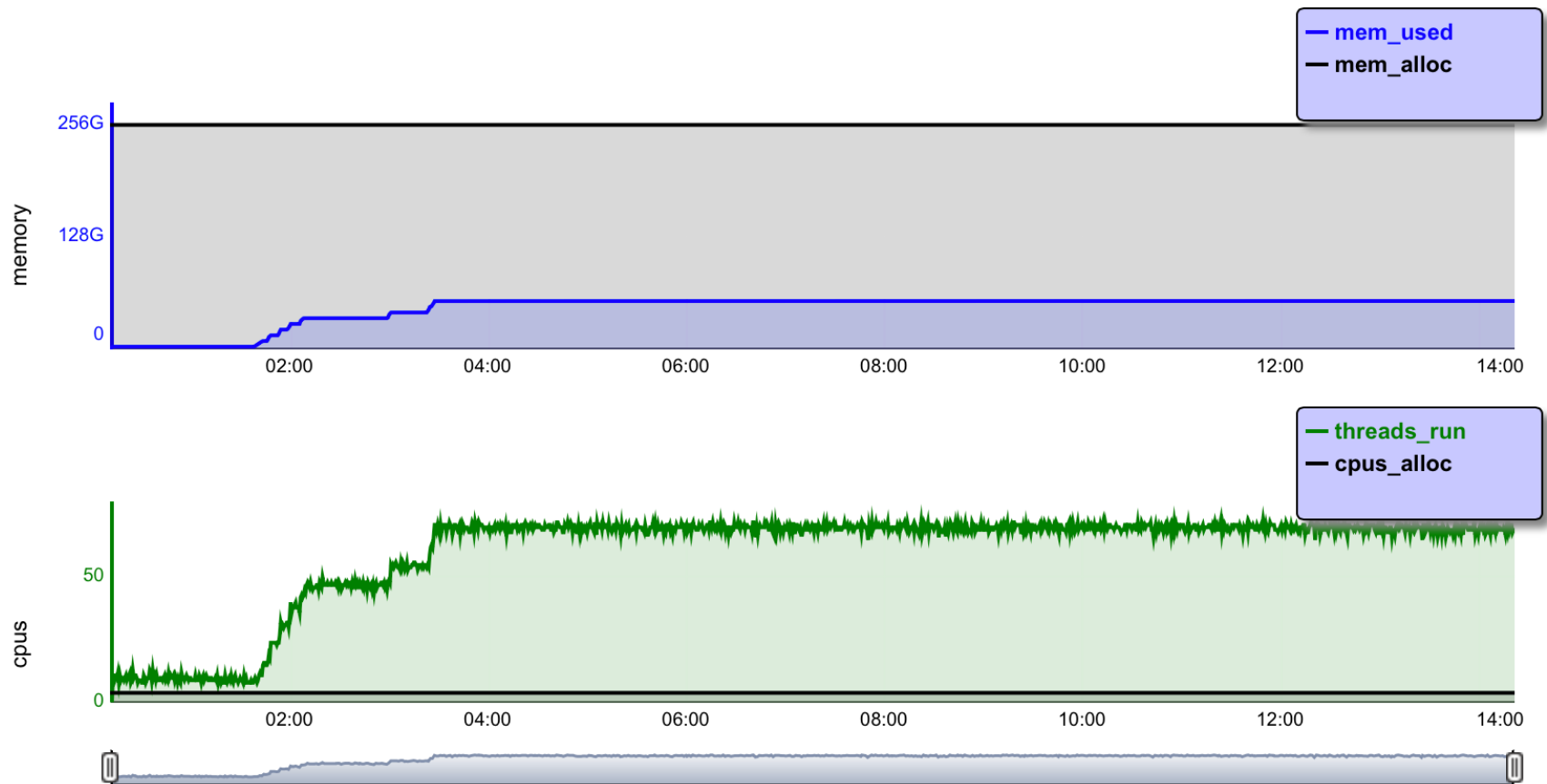
Recommendation: reduce CPU allocation

Another dashboard example



The memory use likely exceeded memory allocation

Recommendation: increase memory allocation to 250 GB. If job completes, determine memory usage with 'jobhist', and then use that for future jobs.



Comment: CPUs badly overloaded

Recommendation: could run in less than half the memory

Hands-on (monitoring)



```
cd /data/$USER/hpc-classes/biowulf/bowtie  
sbatch [--partition=student] --cpus-per-task=16 bowtie.bat
```

- Use ‘`jobload -u $USER`’ to check the status of your running job. How much memory is it using?
- After the job finishes, use ‘`jobhist jobnumber`’ and find out how much memory it used.



Why are my jobs pending?

- 'freen' shows free CPUs but not free memory
- Other jobs have higher priority (sprio)
- Nodes are reserved for higher-priority jobs

```
scontrol show job #####  
squeue -o "%18i %20s"
```

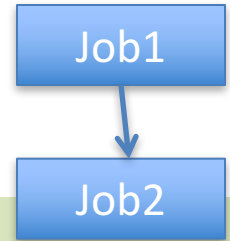
will show you the estimated start time of the job, if known



Suggestions to speed up scheduling of your jobs

- Use `freen` to check cluster status, and consider submitting to a different partition.
- Use the smallest walltime, memory and CPU that your jobs require.
- Submit to multiple partitions
- Use the quick queue (< 4 hr jobs) if possible.

Job Dependencies



```
biowulf% sbatch Job1.bat
```

```
1111
```

```
biowulf% sbatch --depend=afterany:1111 Job2.bat
```

```
1112
```

```
biowulf% sjobs
```

```
[biowulf]$ sjobs
```

User	JobId	JobName	Part	St	Reason	Runtime	Walltime	Dependency	Nodelist
teacher	1111	Job1.bat	norm	R	---	0:17	4:00:00		cn0035
teacher	1112	Job2.bat	norm	PD	Dependency	0:00	4:00:00	afterany:1111	

```
cpus running = 2
```

```
cpus queued = 1
```

```
jobs running = 1
```

```
jobs queued = 1
```

Job Dependencies

<code>--depend=afterany</code>	(after all of the jobs have exited, with or without errors)
<code>afterok</code>	(after all of the jobs have exited with no errors)
<code>afternotok</code>	(after all of the jobs have exited with errors)
<code>after</code>	(after all of the jobs have started)

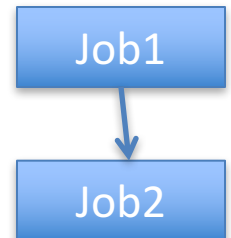
IMPORTANT: Exit status = exit status of shell = exit status of last command

Use 'set -e' in script to exit after first failed command. e.g.

```
#!/bin/bash

set -e

cd /data/$USER/some/dir
...command1...
...command2.....
...command2.....
```



Job Dependencies

Making several jobs dependent on a single job.

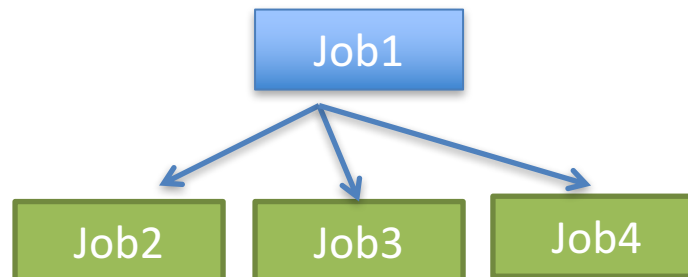
```
biowulf$ sbatch Job1.bat
Or swarm -f swarmfile1
1111

biowulf$ sbatch --depend=afterany:1111 Job2.bat
Or swarm -f swarmfile2 --depend=afterany:1111

1112

biowulf$ sbatch --depend=afterany:1111 Job3.bat
Or swarm -f swarmfile3 --depend=afterany:1111

1113
```



Job Dependencies

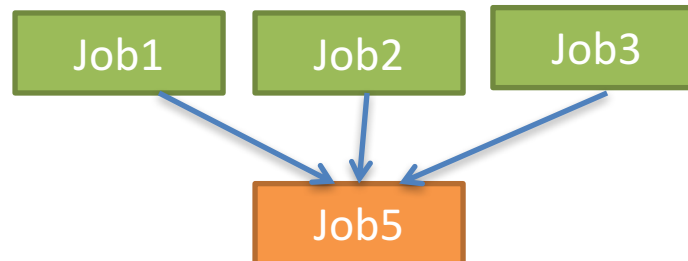
Making one job dependent on several other jobs

```
biowulf% sbatch --depend=afterany:1111:1112:1113 Job3.bat
Or swarm -f swarmfile2 -depend=afterany:1111:1112:1113
1114
```

```
Biowulf% sjobs
```

User	JobId	JobName	Part	St	Reason	Runtime	Walltime	Dependency	Nodelist
teacher	3996129	Job1.bat	norm	R	---	0:17	4:00:00		cn0035
teacher	3996143	Job2.bat	norm	PD	Dependency	0:00	4:00:00	afterany:1111,1112,1113	

```
cpus running = 2
cpus queued = 1
jobs running = 1
jobs queued = 1
```



Job Dependencies

See

https://hpc.nih.gov/docs/job_dependencies.html

for examples of scripting job dependencies in
Bash, Perl, Python.

Hands-on (job dependencies)

```
cd /data/$USER/hpc-classes/biowulf/dependencies
sbatch Job1.bat
```

(select or note down the job number)

```
sbatch --depend=afterany:Job1num Job2.bat
```

(select or note down the job number)

```
sjobs
```

(what is the state of the second job?)

```
sbatch --depend=afterany:Job1num:Job2num Job3.bat
```

```
sjobs
```

(does it show the dependency?)

```
swarm -g 2 --depend=afterany:Job3num -f swarm.cmd
```

Matlab on Biowulf

- Can be multi-threaded.

<http://biowulf.nih.gov/apps/matlab.html>

- License-limited: no Matlab batch jobs on cluster.

http://helixweb.nih.gov/nih/license_status

Compile all Matlab code that you want to run via batch jobs

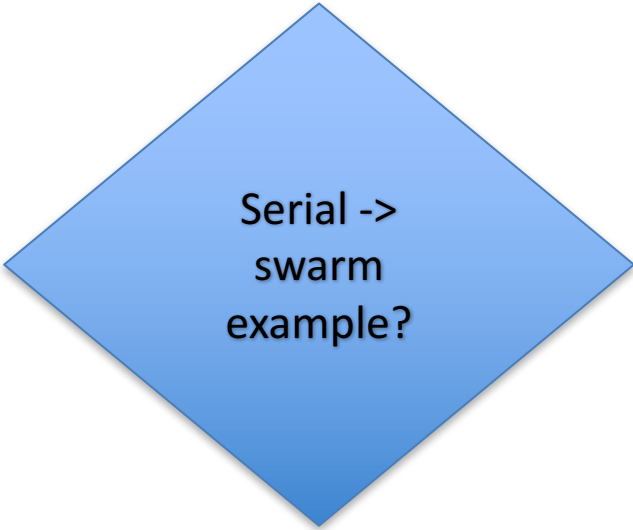
https://hpc.nih.gov/apps/matlab_compiler.html



Decision Point!

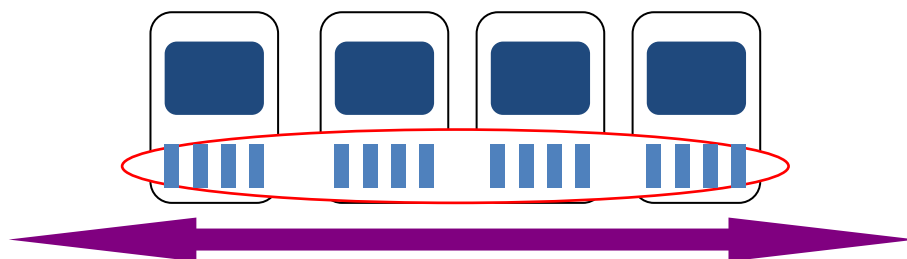


Parallel jobs?



Serial ->
swarm
example?

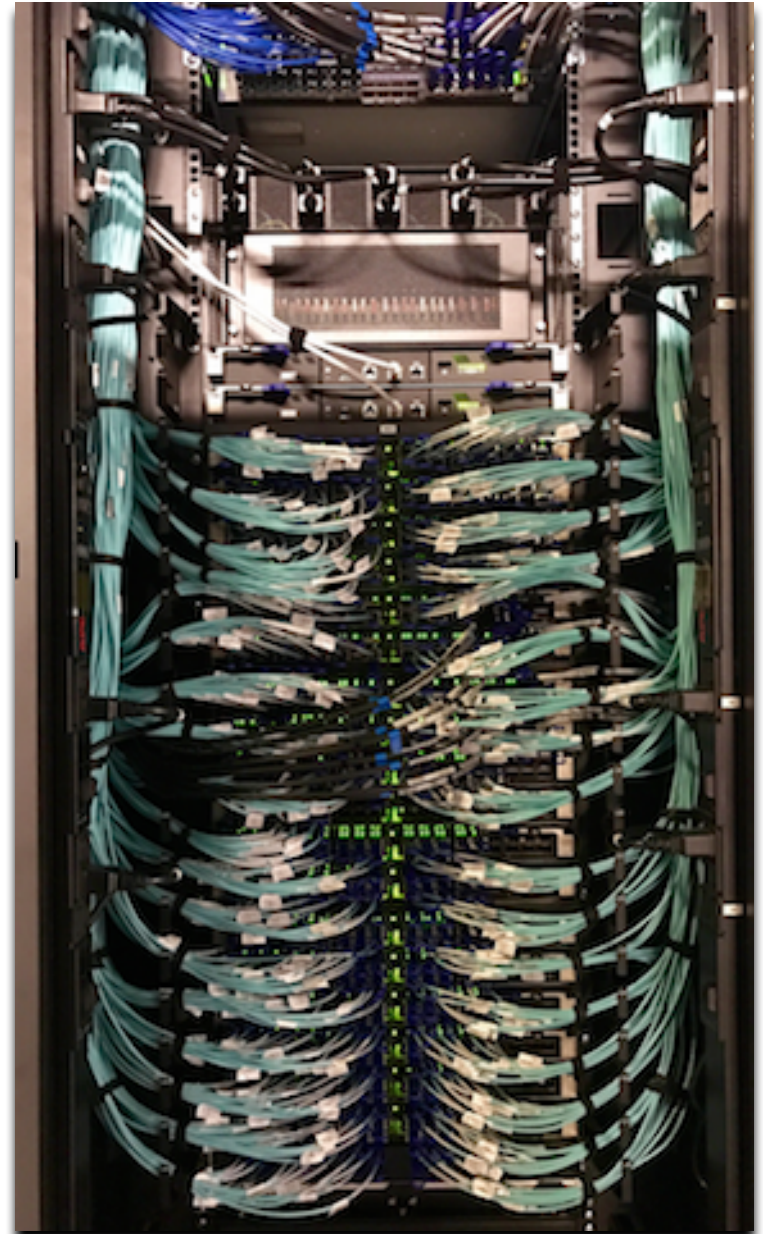
Multi-Node Parallel jobs



- Primarily molecular dynamics (NAMD, CHARMM, etc)
- Distributed memory requires programming/compiling with message-passing libraries (MPI the de facto standard)
- Communication over network requires high-performance interconnects to achieve good scalability

High-performance Infiniband Network

- 100 Gb/s fabric (56 Gb/s to nodes)
- Low latency (more important than bandwidth)
- Bypass TCP/IP stack
- Offloads CPU



~25 km of fiber!



Parallel (MPI) process

Many parallel apps do better with `--ntasks-per-core=1` (ignore hyperthreading)

Required		Command	Allocated	
CPU	Memory		CPU	Memory
C	< 4 GB per MPI process	<code>sbatch --partition=multinode --ntasks=C [--constraint=nodetype] --exclusive --ntasks-per-core=1 jobscript</code>	2 * C (1 per MPI process) (rounded up to nearest 2 if necess.)	4 GB per MPI process
C	G GB per MPI process	<code>sbatch --partition=multinode --ntasks=C [--constraint=nodetype] --exclusive --mem-per-cpu=G --ntasks-per-core=1 jobscript</code>	2 * C (1 per MPI process) (rounded up to nearest 2 if necess.)	2 * G GB per MPI process

Parallel (MPI) process

Use `$SLURM_NTASKS` within the script

```
#!/bin/bash

module load meep/1.2.1/mpi/gige
cd /data/username/mydir

meme mini-drosoph.s -oc meme_out -maxsize 600000 -p $SLURM_NTASKS
```

OpenMPI has been built with Slurm support, so `$np` need not be specified.

```
#!/bin/bash

# this file is called meep.par.bat

# module for ethernet runs
module load meep/1.2.1/mpi/gige

cd /data/$USER/mydir
mpirun `which meep-mpi` bent_waveguide.ctl
```

Infiniband Jobs

All nodes in the 'multinode' partition are connected to FDR Infiniband

```
sbatch --partition=multinode
       --constraint=x2650
       --ntasks=64
       --ntasks-per-core=1
       --exclusive
       jobscript
```

Relion

<https://hpc.nih.gov/apps/relion>

The screenshot displays the RELION v2.0.5 graphical user interface. The window title is 'RELION v2.0.5: /spin1/users/susanc/relion/bench/input'. The top menu bar includes 'File', 'Jobs', and 'Autorun'. Below the menu is a tabbed interface with 'I/O', 'Microscopy', 'CTFFIND', 'CTFFIND4', 'Gctf', and 'Running' tabs. The 'Running' tab is active, showing configuration parameters for a job. On the left, a sidebar lists various processing steps, with 'CTF estimation' highlighted. The main configuration area includes fields for 'Number of MPI procs' (set to 8), 'Submit to queue?' (Yes), 'Queue name' (multinode), 'Queue submit command' (sbatch), 'Local Scratch Disk Space' (200), 'Walltime' (5-00:00:00), 'Memory Per Thread' (8g), 'Standard submission script' (v2.0.6_patched/single_cpu.sh), and 'Minimum dedicated cores per node' (2). There are buttons for 'Print command', 'Schedule', and 'Run now!'. Below the configuration, there are sections for 'Job actions', 'Current job' (Give_alias_here), and 'Display'. The bottom of the window features a large text area for stdout and stderr output, with instructions to double-click to open in separate windows.

RELION v2.0.5: /spin1/users/susanc/relion/bench/input

File Jobs Autorun

I/O Microscopy CTFFIND CTFFIND4 Gctf Running

Import
Motion correction
CTF estimation
Manual picking
Auto-picking
Particle extraction
Particle sorting
Subset selection
2D classification
3D classification
3D auto-refine
Movie refinement
Particle polishing
Mask creation
Join star files
Particle subtraction
Post-processing
Local resolution

Number of MPI procs: 8
Submit to queue? Yes
Queue name: multinode
Queue submit command: sbatch
Local Scratch Disk Space: 200
Walltime: 5-00:00:00
Memory Per Thread: 8g
Standard submission script: v2.0.6_patched/single_cpu.sh
Minimum dedicated cores per node: 2
Additional arguments:

Print command Schedule Run now!

Job actions Current job: Give_alias_here Display:

Finished jobs
CtfFind/job001/

Running jobs
CtfFind/job002/

Scheduled jobs
CtfFind/job003/

Input to this job

Output from this job

stdout will go here; double-click this window to open stdout in a separate window

stderr will go here; double-click this window to open stderr in a separate window

Relion uses MPI and can run multiple threads per MPI task. Recommended: Relion GUI

Upcoming Seminar...

Relion tips and tricks, and Parallel jobs and benchmarking

Date: Jan 16, 1 - 3 pm

Bldg 50, Rm 1227 (next to the coffeeshop)

Mechanics and best practices for submitting RELION jobs to the batch system from both the command line and via the RELION GUI, as well as methods for monitoring and evaluating the results. Scaling of parallel jobs, how to benchmark to make effective use of your allocated resources

Serial -> Swarm : Demo

Mini-project: run Novoalign on each of a group of fastq files to align them against the hg19 human genome

```
[biowulf]$ cd serial2swarm
```

```
[biowulf]$ ls
```

```
fastq_files.tar.gz
```

```
serial.sh
```

```
make_swarm.sh
```

```
make_swarm2.sh
```

Serial -> Swarm : Demo

Factors to consider

- Number of alignments (> 4K?)
- CPUS to allocate for each alignment
- Memory required for each alignment
- Disk space required
- Time taken for each alignment
- Total time

Serial -> Swarm : Demo

First submit the serial job (or run it interactively)

```
[biowulf]$ cd serial2swarm
```

```
[biowulf]$ ls
```

```
fastq_files.tar.gz
```

```
serial.sh
```

```
swarm.sh
```

```
swarm2.sh
```

```
[biowulf]$ sbatch --cpus-per-task=16 --mem=20g serial.sh
```

OR

```
[biowulf]$ sinteractive --cpus-per-task=16 --mem=20g
```

Serial -> Swarm : Demo

Factors to consider...

- Number of alignments (> 4K?) - 90
- CPUS to allocate for each alignment – 16 or 32
- Memory required for each alignment – 8 GB
- Disk space required – $6.6 + 90 * 0.1 \approx 17$ GB
- Time taken for each alignment: 16 min
- Total time: 24 hrs for serial job

Serial -> Swarm : Demo

*Modify the serial.sh script to write out a swarm command file
-> make_swarm.sh*

```
[biowulf]$ sh make_swarm.sh
```

```
[biowulf]$ swarm -f novo_swarm.sh -t 16 -g 10 --time=20:00
```

Total time: ~10 mins for the unpacking, + 15 mins for the swarm subjobs
= ~30 mins

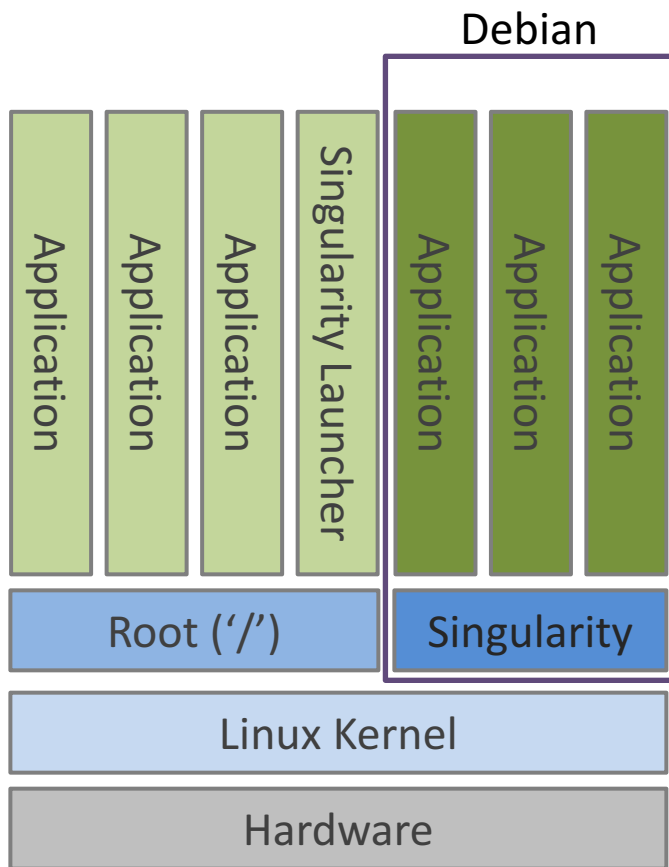
Serial -> Swarm : Demo

A slightly more sophisticated version: `make_swarm2.sh`:

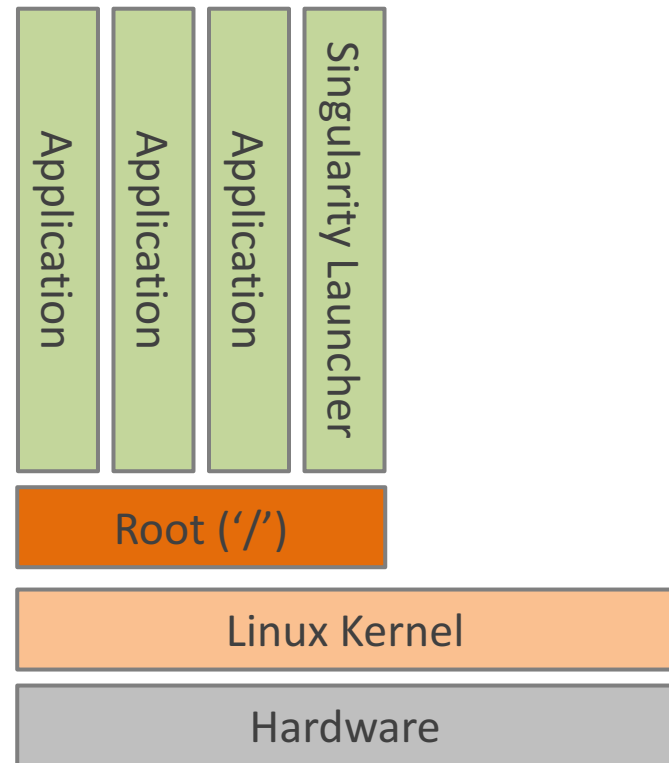
- Unpacks the tar file
- Sets up a swarm command file for the novoalign swarm
- Puts the swarm logs into a subdir
- Sends swarm stderr and stout into a single file per subjob
- Sends an email when the swarm has completed

```
[biowulf]$ sbatch make_swarm2.sh
```

Singularity containers: Mobility of compute

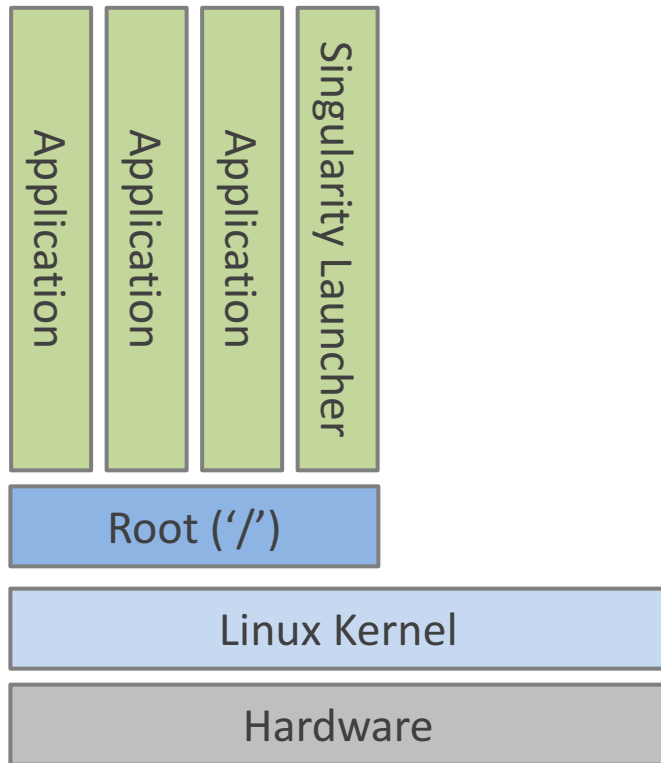


Centos 7 VM

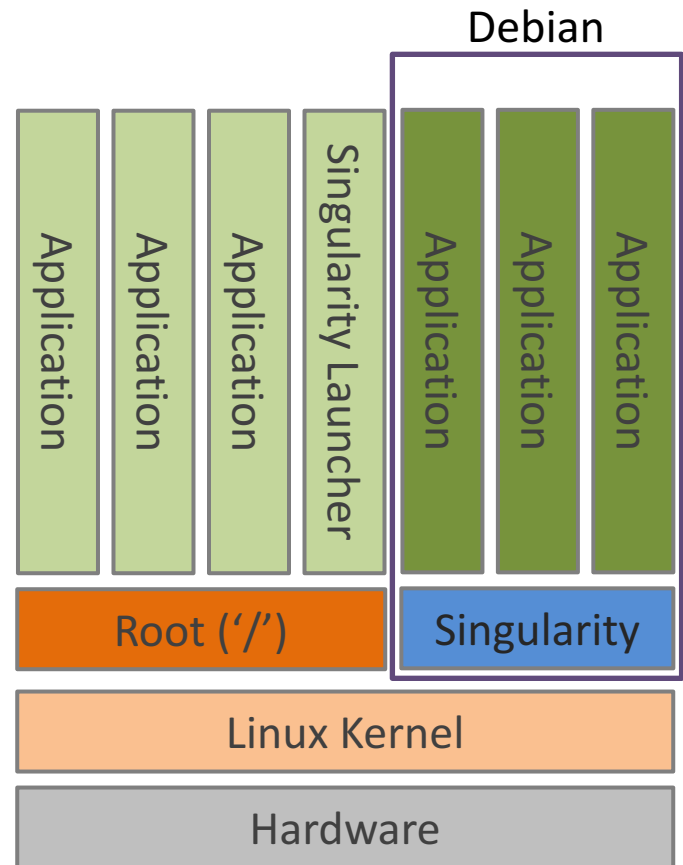


Centos 6 Compute node

Singularity containers: Mobility of compute



Centos 7 VM



Centos 6 Compute node

Singularity Containers: Demo

```
[cloudVM]$ sudo singularity create -s 250 cowsay.app
[cloudVM]$ sudo singularity bootstrap cowsay.app cowsay.def
[cloudVM]$ ./cowsay.app "This is a test"
```

```
< This is a test >
```

```
-----
      \   ^__^
       \  (oo)\_______
          (__)\\       )\/\
              ||----w |
              ||     ||
```

```
[cloudVM]$ scp cowsay.app helix.nih.gov:temp
```

```
[biowulf]$ sinteractive
```

```
[cn1234]$ module load singularity
```

```
[cn1234]$ ./cowsay.app "Running on $(hostname)"
```

```
< Running on cn1234 >
```

```
-----
      \   ^__^
       \  (oo)\_______
          (__)\\       )\/\
              ||----w |
              ||     ||
```

staff@hpc.nih.gov

(aka staff@helix.nih.gov, staff@biowulf.nih.gov)

